MAINTENANCE OF DESIGN AIDS IN A HANDBOOK OF STEEL CONSTRUCTION

by

Babak Ansari

A thesis submitted to
the faculty of Graduate Studies and Research
in partial fulfillment of the
requirements for the degree of

Master of Applied Science

The Master of Applied Science in Civil and Environmental Engineering Program is a
joint program with the University of Ottawa, administrated by the Ottawa-Carleton
Institute for Civil Engineering

Carleton University
Ottawa, Ontario, Canada
May 2003

Copyright © Babak Ansari, 2003

National Library
of Canada

Acquisitions and
Bibliographic Services

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale
du Canada

Acquisitions et
services bibliographiques

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

ISBN: 0-612-83470-0

Our file Notre référence

ISBN: 0-612-83470-0

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

Canada

The undersigned recommend to the Faculty of Graduate Studies and Research acceptance
of the thesis

MAINTENANCE OF DESIGN AIDS IN A HANDBOOK OF STEEL CONSTRUCTION

Submitted by

Babak Ansari, B. Sc.

In partial fulfillment of the requirements of the degree of Master of Applied Science

N. M. Holtz, Thesis Supervisor
Department of Civil and Environmental Engineering
Carleton University

A. O. Abd El Halim, Acting Chair
Department of Civil and Environmental Engineering
Carleton University

Ottawa-Carleton Institute for Civil Engineering

Ottawa, Ontario, Canada

May 2003

Abstract

Design codes and standards undergo continuous review, with significant changes occurring every few years. Changes in these necessitate changes to all corresponding design aids. The objective of the research is to develop techniques that make it possible for computer-based design aids to adapt more readily to code and standard changes.

This thesis has considered clause 13 of the CAN/CSA-S16.1 and some of its design aids in the Handbook of Steel Construction. An easy to modify computer program to automatically generate the Beam Selection Tables was constructed, as were a few other computer aids. Even though only a few clauses of the Canadian steel standard were used as examples, the techniques that were developed are usable for other clauses, codes and standards. The effects of future changes were simulated by applying the changes in the standard and Handbook over the past 20 years.

ACKNOWLEDGEMENTS

Professor Neal M. Holtz at Carleton University supervised this research. His constant encouragement and patience and his assistance in the process of my learning was exemplary. I appreciate all his efforts.

I am thankful to the SLALS (School of Linguistic and Applied Language Studies) at Carleton University. I enjoyed their extensive innovative courses especially the non-credit thesis-writing course, which helped me in the writing of this thesis.

I am grateful for all the great support of the National Spiritual Assembly of Canada and their guidance throughout my studies. They devotedly helped me whenever it was necessary.

I am also grateful to John Buck, my classmate and now a good friend of mine, who offered me whatever he knew with generosity. I appreciate his willingness to help.

For most proofreading and organization of this thesis I am thankful to Kristin Cooper's help and her suggestions. I will never forget her kindness and expertise that was readily shared.

The last but certainly not the least, my appreciation goes to my worldwide scattered family and friends who I am proud of each and every one. I thank my parents who always tried to raise me up an independent individual, who also set high standards for me, while they never stopped loving me. My sister, Nahal, and my brother, Behnam, who supported me a great deal all the time. Moreover, all friends and family who kept me in their minds and prayers, especially Ata, Jaklin, and Vahideh Ansari and Vida Akbarzadch who offered me great vacation time and advice whenever I consulted them and also Ramin Karimi, Armin Rahmanian, Amir Parsa and Elham Seddigh who eagerly and cheerfully provided me with helpful advice and information in computer science field that I needed for my thesis work. I really appreciate their care and friendship.

TABLE OF CONTENTS

<i>ABSTRACT</i>	<i>III</i>
<i>ACKNOWLEDGEMENTS</i>	<i>IV</i>
<i>TABLE OF CONTENTS</i>	<i>V</i>
<i>LIST OF TABLES</i>	<i>IX</i>
<i>LIST OF FIGURES</i>	<i>X</i>
<i>LIST OF ACRONYMS</i>	<i>XI</i>
<i>CHAPTER 1</i>	<i>1</i>
<i>1 INTRODUCTION</i>	<i>1</i>
1.1 Design Standards	1
1.2 Design Aids	2
1.3 Objective	3
1.4 Overall System Structure	4
1.4.1 Scope and Limitation	6
1.5 Thesis Organization	7
<i>CHAPTER 2</i>	<i>9</i>
<i>2 LITERATURE REVIEW</i>	<i>9</i>
2.1 Introduction	9
2.2 Decision Tables	10
2.3 Knowledge-based system environment	13
2.4 Standard Representation Models	14
2.4.1 Text-Based approaches	14
2.4.2 Electronic National Building Code (NBC)	15
2.5 Computerized Design Aids	16
2.5.1 ELRFD	16
<i>CHAPTER 3</i>	<i>17</i>
<i>3 RESEARCH PLAN AND METHODOLOGY</i>	<i>17</i>
3.1 Introduction	17
3.2 Current Maintenance Process of Design Aids	17
3.3 Proposed Maintenance Method	18

3.3.1	Advantages of proposed model	20
3.3.2	Disadvantages of proposed model	21
3.4	Handbook of Steel Construction Format	21
3.5	Case Study	22
3.5.1	Clause 13 of the S16.1 Standard	22
3.5.2	The Beam Selection Tables	23
3.5.3	The Beam Load Tables	25
3.5.4	The Class Number Tables	25
3.6	Test Plan	25
3.7	Evaluation Method	26
3.8	Summary	26
<i>CHAPTER 4</i>		<i>27</i>
<i>4 SOFTWARE DEVELOPMENT AND PROGRAMMING TECHNOLOGIES DETAILS</i>		<i>27</i>
4.1	Introduction	27
4.2	Object-Oriented Programming (OOP)	28
4.2.1	Python	29
4.3	Web-based Design Aids	29
4.3.1	CGI (Common Gateway Interface)	30
4.3.2	Server feasibility	31
4.3.3	Running on all platforms	31
4.3.4	Installation and upgrading	31
4.3.5	XML	32
4.3.6	CVS (Concurrent Versions System)	33
4.4	Pydoc Module	34
4.5	Implementation of Technology	35
4.5.1	Variable classification	35
4.5.2	Class and methods and clauses	36
4.5.3	Text-based programming	36
4.5.4	Data Orientated Style	37
4.5.5	Dictionary	38
4.5.6	Query	39
4.5.7	Data identifier	40
<i>CHAPTER 5</i>		<i>42</i>
<i>5 OVERVIEW OF SOFTWARE ARCHITECHTURE</i>		<i>42</i>
5.1	Introduction	42
5.2	Beam Selection Tables	42
5.2.1	Module structures	42
5.2.2	Logical Interaction between modules	45

5.3	Beam Load Tables	52
5.3.1	Module structures	52
5.3.2	Logical interaction between modules	52
5.4	Class of Sections in Bending	53
5.4.1	Module structures	53
5.4.2	Logical Interaction between modules	53
<i>CHAPTER 6</i>		<i>54</i>
<i>6 SOFTWARE DETAILS AND DOCUMENTATION</i>		<i>54</i>
6.1	Introduction	54
6.2	Documentation	54
6.2.1	Standard S16.1-M87	54
6.2.2	Beam Selection Tables	57
6.2.3	Beam Load Tables	58
6.2.4	Class of Sections in bending	60
6.2.5	Design Parameters	61
6.2.6	Member Segment	62
6.2.7	Steel Properties	64
6.2.8	XML Table	76
6.2.9	Utils	77
6.2.10	IShape	77
<i>CHAPTER 7</i>		<i>80</i>
<i>7 EXAMPLES AND EVALUATION</i>		<i>80</i>
7.1	Introduction	80
7.2	Testing	80
7.2.1	Module tests	80
7.2.2	Package tests	81
7.3	Evaluation	81
7.3.1	Source code changes measurement	81
7.3.2	Scalable modules	83
7.3.3	Easy to plug modules	83
7.4	Summary	84
<i>CHAPTER 8</i>		<i>85</i>
<i>8 SUMMARY AND CONCLUSIONS</i>		<i>85</i>
8.1	Summary	85
8.2	Conclusion	87
8.3	Future Research	87
<i>REFERENCES</i>		<i>89</i>
<i>APPENDIX A</i>		<i>94</i>

Source codes	94
Standard S16.1-M87	94
Beam Selection Tables	100
Beam Load Tables	102
Class of Sections in bending	104
Design Parameters	106
Member Segment	108
Steel Properties	109
XML Table	115
Utils	117
IShape	119

LIST OF TABLES

<i>Table 2.1. Hierarchic organization of NBC</i>	<i>15</i>
<i>Table 7.1. Required Changes to Obtain Fifth Edition of The Handbook from the Fourth edition</i>	<i>82</i>

LIST OF FIGURES

<i>Figure 1.1. Scope of present research</i>	<i>5</i>
<i>Figure 3.1. Existing Handbook modification process</i>	<i>18</i>
<i>Figure 3.2. Proposed Handbook modification process</i>	<i>19</i>
<i>Figure 5.1. Positive directions of the loading P and deflection y.....</i>	<i>44</i>
<i>Figure 5.2. Positive direction of the bending moment M</i>	<i>44</i>
<i>Figure 5.3. Positive direction of the shear force V.....</i>	<i>44</i>
<i>Figure 5.4. Package diagram</i>	<i>46</i>
<i>Figure 5.5. Class diagram</i>	<i>48</i>
<i>Figure 5.6. Sequence diagram.....</i>	<i>51</i>

LIST OF ACRONYMS

AASHTO	American Association of State Highway and Transportation Officials
API	Application Program Interface
ASTM	American Society for Testing and Materials
BLT	Beam Load Tables
BST	Beam Selection Tables
CAE	Computer-Aided Engineering
CAN/CSA-S16.1	Limit State Design of Steel Structure (by Canadian Standards Association)
CD-ROM	Compact Disc Read-Only Memory
CGI	Common Gateway Interface
CISC	Canadian Institute of Steel Construction
CST	Class of Section in Bending Table
CVS	Concurrent Versions System
DOS	Data Orientated Style
ELRFD	ELRFD Electronic Load and Resistance Factor Design
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
NBC	National Building Code of Canada
NBS	National Building Standard (American)
OOP	Object-Oriented Programming
OS	Operating System
PC	Personal Computer
S16-M78	Canadian Steel Design Standard (by Canadian Standards Association)
SASE	Standards Analysis, Synthesis and Expression
SGML	Standard Generalized Markup Language
SICAD	Standards Interface for Computer-Aided Design
SPEX	Standards Processing Expert
SQL	Structured Query Language
SST	Steel Shape Tables
SST	Structural Shape Table
TLC	Tool Command Language
UML	Unified Modeling Language
WWW	World Wide Web
XML	Extensible Markup Language

CHAPTER 1

1 INTRODUCTION

The objective of this thesis is to develop techniques that make it possible for computer-based design aids to adapt more readily to changes in design codes and standards.

1.1 Design Standards

Structural engineering design is governed by a number of codes and standards. For example, an early step in structural design is to determine the applied loads on a structure. To do this, the engineer refers to the appropriate building code, which depends on where the structure is located. For a structure anywhere in Ontario, an engineer has to use the Ontario Building Code [1] to determine loads and forces. To design or size the structural members, the engineer must use another standard that depends on the material of the member as well as the location of the structure. For example, to design a steel beam for a building in Ottawa or anywhere in Canada, the engineer would refer to CAN/CSA-S16.1-94 [2], which is a national standard in Canada for the design of steel structures.

These building codes, referred to in this thesis as “standards” to differentiate them from computer codes, undergo periodic changes. Minor changes such as corrections are issued as needed –sometimes more than once per year. More substantial changes are made in new editions and these come out at roughly five-year intervals.

1.2 Design Aids

A variety of design aids have been created to assist the structural engineer. These traditionally have been paper-based but increasingly are computer-based. One set of paper-based design aids is the Handbook of Steel Construction [3] (which is referred to as “Handbook” in the thesis) and is chosen as the case study for this thesis.

The Handbook has a number of tables that help the engineer in calculations; for instance, the Beam Selection Tables provide a useful tool that decreases the amount of manual calculations required for beam selection. Without the Beam Selection Tables, the designer has to choose a section, try its adequacy and repeat this process for each section to ensure that the selected beam is a safe and optimum one.

Each design aid is based at least partially on the standard. When the standard changes, the design aids must also change. The Handbook has been revised often due to rapid changes of the steel standard in the past decade, as well as to changes in properties and availability of structural steel. In addition, changes in layouts and contents of tables are made frequently and independently from changes in the standard. In the past 20-

years, there have been 6 distinct steel design standards and 7 different editions of the Handbook.

1.3 Objective

Design standards undergo continuous review, with significant changes occurring every few years. Changes in these necessitate changes to those design aids that are based on the standards. These changes to the design aids require significant effort, which can be very costly. Performing the changes is time-consuming, so that by the time the revised design aids have been updated, the standard on which they are based may be obsolete or nearly so. In addition, this means that design aid software may be considerably behind the standard in day-to-day practice.

This thesis has considered clause 13 (Member and Connection Resistance) of the CAN/CSA-S16.1 [2] and its design aids in the Handbook [3] as a case study. The programs developed for this thesis generate the Beam Selection Tables (BST), Beam Load Tables (BLT) and Class Section in Bending Tables (CST) with more features than are currently in the handbook. They are suitable for generating the text of the handbook as well as for interactive design use.

A secondary objective of this thesis is to explore modern techniques of software development, including cooperative programming, programming languages and source code control systems. All of these techniques contributed to making software easier to maintain.

The modifiability of software was tested by first implementing the programs appropriate to Edition 3rd (1980) of the Handbook, and then applying all the changes in stages to finally reach the software for Edition 7th of 2000. In effect, we studied the evolution of the software over a 20-year time period.

A final objective was to develop methods that could be used to track the effort required to modify software over a long time period. These methods were then used to evaluate a software development technique that is claimed to lead to more maintainable programs.

The primary objective of this thesis was to develop a computerized version of the Handbook that easily adapts to changes in the standard.

1.4 Overall System Structure

This section describes the overall structure of a proposed system for the computer based representation and use of design standards. It also describes the scope of the present thesis and its place within the entire project.

The overall system design is shown in Figure 1.3. The major ideas of the model are these:

1. The design standard (such as S16.1) will reside in a “Standard Database”. That Standard will be in a presentation- and programming language- neutral form. The database will support standard word processing operations so it could be used as the master database to support editing by the technical committee, but that particular aspect is outside the scope of this research.
2. Design aid software will be developed to generate design tables for the Handbook, and to provide interactive design aids for the engineer. This software maybe developed in a variety of programming languages. Fortran, Visual Basic, C, C++, Java and Python will perhaps be the most common, although in principle any programming language may be used.
3. When the design aid software needs to perform a computation that depends on logic specified by the design standard, it queries the standard database for an implementation of that logic, and specifies the programming language that is in use.

Figure 1.1. Scope of present research

The database returns something that can be used to perform the computation. For example, to a C++, Java or Python program, the database will return an object that can be “called” and that will return the desired results.

Applications can query for other things as well, including text for display, data requirements for a computation, etc. Again, the details of this are beyond the scope of the present investigation.

1.4.1 Scope and Limitation

The scope of the present investigation is suggested by the dashed outline in Figure 1.3. The more precise details are these:

1. The design of the major application program interface (API) between the Design aid software and the database. This is shown as the Query API on the figure.
2. The development of a small number of design aids that can be used as a test bed for the development of the database.
3. The use and description of a software development environment that eases the task of managing cooperative development among many people.
4. The development of a methodology to evaluate how software changes over time. This allows actual measurement of the effort required to modify software and will be used to test the validity of ideas developed by the whole project.

This latter item may be the most significant for the entire project, but all taken together, the current thesis has developed the necessary infrastructure that will enable the

completion of the entire project. A companion thesis will develop the internal details necessary for overall success.

This case study is limited to the S16.1 steel standard [2]. However, most of the modification techniques developed here can also be used for other design standards. In general, a programmer will find the techniques simple and practical when modifications of computer packages are required.

1.5 Thesis Organization

Chapter 2, LITERATURE REVIEW, is a review of the existing ideas and research in the literature regarding developing design aids and in particular, computer aided design packages.

Chapter 3, RESEARCH PLAN AND METHODOLOGY, is descriptions of research, its need, and the justification of the technology that should be used in the software structure. The case study chosen for this thesis and its related design aids is presented.

Chapter 4, SOFTWARE DEVELOPMENT AND PROGRAMMING TECHNOLOGIES DETAILS, lists techniques that are used to develop programs with a brief introduction of these techniques. Advantages and disadvantages and in which capacity each one serves maintenance is covered.

Chapter 5, OVERVIEW OF SOFTWARE ARCHITECTURE, discusses and illustrates the high-level design of the developed software. The illustration shows the general structure and interaction between different parts of the software.

Chapter 6, SOFTWARE DETAILS AND DOCUMENTATION, presents the details of the program, with full documentation.

Chapter 7, EXAMPLES AND EVALUATIONS, presents an evaluation of the effectiveness of implemented maintenance techniques. A test plan and its results are presented.

Chapter 8, CONCLUSION, summarizes the technologies used, gives specific concrete conclusions of achievements and suggests themes for future research.

2 LITERATURE REVIEW

Most previous research has concentrated on the development of computer-based representations of design codes and standards. While it was always implicitly recognized that a major benefit would have been to ease the maintenance of design software, the maintenance issue was rarely made an explicit part of the research.

2.1 Introduction

To deal with the numerous calculations involved in the design process, a number of studies have attempted to help the designer in various ways. Studies were focused on topics such as organizing the logic of the standard, and on the representation of standards. A review of significant research is presented in this chapter.

Along with above-mentioned research, computing technology has improved and made a significant contribution to design aids. A review of design aids technologies is also presented in this chapter.

2.2 Decision Tables

Fenves in 1966 [4] defined logic in computer applications as “a chain of reasoning” and referred to their secondary role--after primarily numerical or computational roles—in engineering problems. The logical relationships in design standards were demonstrated by using tabular decision logic and then decision tables converted to flow diagrams, and, thus into computer programs. Fenves applied the method of decision tables for formulating complex decision processes. It was one of the first techniques to formulate, display and document the decision process. Application of this method required translating engineering decisions into rigorous sets of decision tables, and this was not a straightforward mechanical process. However, it has been regarded as a simple approach of using the computer for processing the logic and served as flowcharting aids to be used with conventional procedural languages.

Fenves, along with Goel [5], developed algorithms based on conditional execution and recursive use of data. The algorithms provided an approach that was independent of design constraints as the data were in the form of decision tables. The limitation lies in that the necessary logics for the program were not automatically derived from the decision tables. However, they developed a checking algorithm that provided the ability to change one or two parameters and test the design again. Interestingly enough constraints were provided to the program as data, input as decision logic tables.

In 1975 Nyman and Fenves [6] suggested their model, which was based on tabular decision logic. They categorized design specifications into logical structures of design criteria and function data. The four-level model was hierarchical, consisting of major divisions corresponding to the type of structural components, subdivisions consisting of stress states and limit states and their subdivision had details of components types. The development of design checking programs based on this model was discussed but the programs were not produced.

Studies directed toward assisting standards writers in authoring design standards that essentially retained the four-level approach were conducted by Fenves [7], Fenves and Wright [8] and Harris and Wright [9]. The result of this series of studies was a software system called SASE (Standards Analysis, Synthesis and Expression) developed by the NBS [10]. SASE provided tools for creating and checking decision tables, information networks, classification systems, and organizations of provisions of standards. SASE also provided standards writers with the ability to develop and manage alternative or successive versions of a standard.

The motivation of avoiding the need to recode design programs was first emphasized by Rehak [11], who proposed a CAE (computer-aided engineering) system, which would include a generic design standards processor operating on a formal representation of the governing design standard. In this system, changes in a design standard would only require changes in data.

The development of the SICAD (Standards Interface for Computer-Aided Design) system was initiated by Lopez and Elam [12] in the mid 1980's. This was a software prototype developed to demonstrate the checking of designed components as described in application program databases for conformance with design standards in the SASE representation. The prototype implemented a user interface, along with three standards processors:

- The *organizational network processor* determines which provisions of a standard are applicable in a given context.
- The *information network processor* evaluates, in a given context, each of the provisions selected by the user from the list of applicable provisions, using the *mapping processor*.
- The *mapping processor* evaluates the data item requested by the *information network processor* using mapping functions into the application program database; if no mapping is found, the user is prompted to assign a value to the data item.

The underlying concepts of the SICAD system are in use today in the AASHTO Bridge Design System, 1994.

In 1987, the SPEX (Standards Processing Expert) system was introduced by Garrett and Fenves [13]. This system used a standard-independent approach for sizing and proportioning structural member cross-sections. The system interacted with the

model of a design standard, represented using the four-level SASE representation, to create a set of constraints on a set of basic data items that represented the attributes of a design to be determined. SPEX implemented a standard-independent design process in that both the process of generating the set of constraints from the standard and the process for finding the optimal solution of these constraints, through a numeric optimization system, were generic, and thus not specific to the design standard being used.

2.3 Knowledge-based system environment

Rasdorf and Wang [14] studied representation of standards in a knowledge-based expert system environment. They claim that processing in a knowledge-based expert environment is flexible in checking conformance of design, effect of changing design variables and to evaluate the impact of downstream activities and events. They also introduce the idea of developing standards as databases. They validate design tables that ensure completeness and clarity of standard. Rasdorf integrated the representation of decision tables and information networks using “provisional and organizational facts”, which is very similar to decision tables and requires a manual translation from text of standard to decision tables. This translation requires significant effort and is error-prone and hard to check due to the fact that the translated version is very different from the text of the standard.

Several approaches for representing design standards and for design aid purpose have been proposed and implemented using different underlying concepts and technologies. These approaches will be reviewed in the sections that follow.

2.4 Standard Representation Models

Different approaches with several models of computer applications for building standards has been presented [15], [16]. Text-Based approaches, and advantages and disadvantages will be discussed in this section due to its relevance to the research.

2.4.1 Text-Based approaches

2.4.1.1 Hypertext-Based

Hypertext has been introduced for delivery of standards as a powerful tool. Hypertext as a medium to present the standard has been proposed by Cornick [17]. A prototype of hypertext standard was developed by Malasri [18]. Yabuki and Law [19] presented a model to serve large heterogeneous document storage.

Williams [20] presented the design of a Standards' Document Server that provides client application with displayable information, formulae and table lookups of design standards. This allows the client application to adapt to the latest changes to the standard by being able to access the current information using a well defined interface to the server and thus to the standard.

Frattaroli [21] developed an interactive web based version of CAN/CSA-S16.1-94 called Hyper-handbook. Hyper-handbook provides an interactive, and easily navigable version of CAN/CSA-S16.1-94. Calculator tools, customized table generators, and section tables with descriptive photos were features of the Hyper-handbook.

2.4.1.2 CD-ROM and internet-based approaches

Turk and Vanier [22] also concentrated on representation of standards. According to these authors on-line distribution codes have the advantages of just in time delivery, source sharing, unlimited capacity and a shorter distribution time lag of versus the CD-ROM format. At the time of writing their work, HTML did not support tables, indices and formulae. SGML was suggested so that it could act as a neutral medium and applications could be easily adapted to web technologies.

2.4.2 Electronic National Building Code (NBC)

Cornick [17] categorized the National Building Code of Canada to the following hierarchy:

Table 2.1. Hierarchic organization of NBC

Numbering	Category
13	Part
13.4	Section
13.4.1	Subsection
13.4.1.1	Article
13.4.1.1. (1)	Sentence
13.4.1.1. (1). a	Clause
13.4.1.1. (1). a. (i)	Sub-clause

The hierarchic organization of the NBC makes it easy to represent its contents utilizing hypertext techniques. The National Research Council of Canada and some private firms developed an electronic version of the NBC called "HyperCode". The electronic NBC provides visual information to the designer. However, mathematical formulae and tables of HyperCode need to be presented in a more user friendly way. Also, the user is responsible to use an updated printed version of the NBC document.

2.5 Computerized Design Aids

2.5.1 ELRFD

ELRFD (Electronic Load and Resistance Factor Design), an interactive PC-based tool for conformance checking of structural elements has been developed in a previous study [23]. A chain of linked decision tables corresponding to each design requirement is evaluated. The result is a report of satisfied or unsatisfied design conditions and a summary of the process that has been followed in the design check. The report is an editable text that the user can interact with by inputting design attributes.

Numerous technologies that are employed for design aids fail to consider the issue of design aid maintenance.

CHAPTER 3

3 RESEARCH PLAN AND METHODOLOGY

This chapter provides a justification of ideas for the development of computerized design aids and reviews the methods that meet the requirements. A proposal for a test plan is also provided.

3.1 Introduction

In order to understand how to update steel design aids, the current maintenance process is studied. Software that generates the same design aids as in the Handbook is proposed. How the software will be able to accommodate changes and lend itself to future web-oriented software is discussed. Also, a software test and evaluation plan and case study is discussed in the following sections.

3.2 Current Maintenance Process of Design Aids

The paper Handbook requires changes when changes are made to steel standards. The figure 3.1 illustrates how the changes would occur with a paper document.

Inevitably, this process would take considerable time. The figure 3.1 illustrates how the Editing committee would first decide what modification should apply to the standard. The changes would then be processed, edited and published in the paper document as the next edition of the Handbook. Then a programmer manually extracts formula, orders statements and translates them in order to write computer program code.

Figure 3.1. Existing Handbook modification process

3.3 Proposed Maintenance Method

A new process is proposed to automate the modifications of design aids software as much as possible. Details are shown in the figure 3.2.

As the figure 3.2 illustrates, the editing committee would first decide what changes should be applied to the standard. The changes would be performed on an electronic version of the standard stored in a database. This also gives the option to publish the standard as a paper document.

When a new version of the Handbook is published, it requires updated design aids. The standards database would supply formula, statements and logic to the program code. Based on the new standard, software that is dependent on the standard will generate design aids as well as teaching aids. Details of actualizing this step of the process and the programming details are explained in the following chapters.

Figure 3.2. Proposed Handbook modification process

The design standard database will include the previous standard as well as any new changes that the editing committee would like to apply. Eventually, a semi-automatic process will regenerate the applications design aids as well as teaching aids. This work presents methods, which contribute to the achievement of this automation as much as possible. Development of the design standard database is beyond the scope of this thesis.

3.3.1 Advantages of proposed model

There are many advantages to the model as listed below:

- ❑ *Editing features*, Computerized design aids are easier to edit than paper copies, therefore the cost factor is obviously less than the paper version.
- ❑ *Interactive tables*, Unlike the design aid tables in a paper version, a designer will be able to create tables of his/her own. For example, the designer could generate a table with any arbitrary column-title value needed.
- ❑ *Sorted tables*, Tables could be sorted by shape properties, member type or any behavior such as limit state capacity, etc.
- ❑ *One-stop shopping*, There are a few different Standards that are used in the design process. For example the local load, steel, and concrete Standard are used for the design of composite elements. A web-based Hyper-handbook enables the user to have all Standards on a server. Users would be able to access them just by clicking on them.

After using the Standard and getting the result, the result should be transferred to use as an input for another Standard. For example, to design a

composite section in Canada, applied loads should be obtained by using the Canadian Building Code. The result should be transferred when the designer uses a steel and concrete Standard. Automating this step is feasible by using web-based design aids. This automation fosters faster design processes and eliminates manual errors.

- *Extra features*, Not only can the user customize the design aids, but also could add extra features to it to enhance it. Companies who deal with special applications and design methods could hook their application to the design aid and get special features that are appropriate for their usage.

3.3.2 Disadvantages of proposed model

Computerized design aids need their own infrastructure. Obviously, the more they offer the more complicated an infrastructure is needed.

3.4 Handbook of Steel Construction Format

With current advances and continuing development in web technology, it is possible to create and present versions of hyper documents. Hyper documents are documents in which text or images are linked to other relevant part of the document. Hypertext format makes software connectable to the hyper-handbook or any future web-based handbook. Also a hypertext format of design aid could be printed and used as a paper version of the Handbook. Moreover, it could be distributed as a CD-ROM. The proposed software will publish the design aids in a hypertext format.

3.5 Case Study

The CISC Handbook of Steel Design is chosen for this research, as the underlying standard is relatively simple. In addition, there have been extensive changes to it during the past 20 years and maintenance of its design aids is a matter of interest both from an academic perspective as well as for industry concerns.

3.5.1 Clause 13 of the S16.1 Standard

Clause 13 determines the strength requirements of members and the resistance requirements of connections. Shear strength requirements are presented in sub-clause 13.4. Bending strength requirements are presented in sub-clause 13.5 for laterally supported members and in 13.6 for laterally unsupported members.

For this study, we have chosen to concentrate on clause 13 of the S16.1 Standard and three related design aids from the Handbook: the Beam Selection Tables, the Beam Load Tables and the Class Numbers Tables.

In S16.1, clause 13 is the one that has more parameters than others and therefore interacts with a lot of design aids in the Handbook. In another words most of the design aids in the Handbook have a connection to it. This clause for its unique role is chosen as a case study.

3.5.2 The Beam Selection Tables

The Beam Selection Tables (BST) is a design aid that has been changed in all editions of the Handbook and deals with a number of parameters more than any other one. This has raised an essential need for an effective strategy for its maintenance. Also it largely uses clause 13 of the steel standard.

The Beam Selection Tables is described in the 7th edition of the Handbook as follows:

“The beam selection tables list beam sizes in descending order of their factored moment resistance M_r (shown in bold) based on full lateral support (Clause 13.5, *CAN/CSA S16.1-94*). Listed beams include all WWF 2000 to WWF 700 sizes, W shapes normally used as beams, and all Canadian C shapes. Tables for WWF sizes are based on $F_y = 350$ MPa for G40.21 grade 350W, while those for W shapes are based on $F_y = 345$ MPa, corresponding to the least value among the 3 grades represented (G40.21 350W, ASTM A992 and A572 grade 50). Tables for C shapes are based on G40.21 300W with $F_y = 300$ MPa. Tables from earlier editions for WWF sizes and W shapes based on G40.21 300W material are now in Part Eight of the Handbook. A table for S shapes based on ASTM A572 grade 50 steel with $F_y = 345$ MPa is also included.

Shapes shown in bold type are generally the economy sections, based on both M_r and mass. Other shapes listed below a bold-type section are heavier sections, but may be economy sections when depth limitations require a shallower beam, or when shear resistance of a coped beam influences the beam selection.... Guidance as to factored shear resistance of coped beams is provided in the Beam Load Tables....

For each beam size, the tables list the maximum unsupported length L_u for which the factored moment resistance M_r is applicable. In addition the tables list the factored moment resistance M_r' for laterally unsupported

beams (Clause 13.6, S16.1-94) for selected values of the unbraced beam length greater than L_u . For other values of unbraced length greater than L_u , M_r' can be interpolated.

The following items are included in the table:

V_r = factored shear resistance (kN)

= $\phi A_w F_s$ (Clause 13.4.1.1, S16.1-94)

I_x = moment of inertia about x-x axis (10^6 mm^4)

b = flange width (mm)

L_u = maximum unsupported beam length for which M_r is applicable (mm)

M_r = factored moment resistance for laterally supported member (kN.m)

= $\phi Z F_y$ for Class 1 and Class 2 sections

= $\phi S F_y$ for Class 3 sections (Clause 13.5, S16.1-94)

M_r' = factored moment resistance for tabulated unbraced beam length when greater than L_u (kN.m)

(M_r' is computed according to Clause 13.6 of S16.1-94 using $\omega_2 = 1.0$ in the expression for M_u .)"

and follows by a description of how to use the BST as a design aid:

"Compute the maximum bending moment in the beam under factored loads M_f and the required moment of inertia I_{reqd} to meet the deflection limit using the specified loads.

For a laterally supported beam, proceed up the M_r column until a Value of $M_r > M_f$ is obtained. Any beam above will satisfy the factored moment requirement. Check to ensure that $V_r > V_f$, the maximum factored beam shear, that $I_x > I_{reqd}$, that L_u is greater than the maximum unsupported beam length.

For a laterally unsupported beam, proceed up the M_r column until a value of $M_r > M_f$ is reached. Then move to the right across the table to the column headed by the unsupported length (or the first listed unsupported length greater than required) to obtain a value of M_r' . Proceed up this column comparing a few beams have an $M_r' > M_f$ and choose the lightest section. Check $V_r > V_f$ and $I_x > I_{reqd}$ if a deflection check is necessary."

The BST (Beam Selection Tables) prepared by the software is structured and presented similar to the BST in the Handbook. Therefore it not only could offer all design aids that the Handbook offers, but it also has the advantage of a computerized design aid.

3.5.3 The Beam Load Tables

Beam Load Tables (BLT) lists total uniformly distributed factored loads with the same condition that BST offers. BLT is structured the same way that BST are in the Handbook.

3.5.4 The Class Number Tables

Class Section in Bending Tables (CST) lists the class of section in bending of WWF sizes and W shapes for four common grades of steel.

3.6 Test Plan

A test is planned here to determine how well the software developed in this research lends itself to modification of design aids. In this test, the software is modified to generate design aids exactly as they appear in the 3rd, 4th, 5th, 6th, 7th editions and the proposed 8th edition of the Handbook. The proposed 8th edition of the Handbook accommodates all changes that, at the time of writing this thesis, needed to be applied to the design aids in order to match the current edition of the steel standard.

3.7 Evaluation Method

The software is modified to generate design aids that are exactly as they appear in the corresponding edition of the Handbook. Each modified copy of the software is referred to as a *version* of the software and is labeled with its corresponding Handbook edition number. The effectiveness of the software methodology is evaluated by determining differences between different versions of the Handbook and comparing with the differences in the software.

3.8 Summary

A new process is proposed to ease generating design aids and software to give more features to the user. A solid test and evaluation plan ensures that the proposed software lends itself to the modifications of design aids.

During planning it is considered that a user of the software would have the option of getting a paper version, or a CD-ROM format of the Handbook,

A web-based version of the software is not in line with the research topic and is not going to be developed. For the development of the web-based version, all required needs for its infrastructure such as being platform independent should be foreseen. An XML presentation of the result is desirable.

CHAPTER 4

4 SOFTWARE DEVELOPMENT AND PROGRAMMING TECHNOLOGIES DETAILS

This chapter introduces the technologies that are used in the development of the software and suggests how they should be used to achieve the easier maintenance goal.

4.1 Introduction

Some existing technologies are considered in developing programs for the design aids in the Handbook. These technologies have contributed to the achievement of techniques for automatic modification of design aids. The following outlines a brief introduction to these technologies and reviews how they are used.

4.2 Object-Oriented Programming (OOP)

Object-Oriented Programming is a well-established methodology in programming that offers unique features. Two of these features suit the development of design aid programs:

(1) The variables in each class are separated from each other. To understand the class behaviour we could picture a class as a box. Variables of different classes do not interact, in the same way that the contents of boxes do not mix with those of other boxes. This feature plays an essential role in maintenance of a program, as there is no need to trace through all the program code after changing a variable's name or value. Modification of a program in each class is independent and does not affect any other part of the program. This leads to easier maintenance of programs.

(2) A class in OOP acts very similar to a black box that includes methods. By sending messages to the black box another message is returned. By changing the content of box the returned message will be changed but the methods that were working with the black box still work. In this way when a method changes the structure of the software does not change. This means less changes in the source code of the software.

4.2.1 Python

Python combines remarkable power with very clear syntax. It has modules, classes, exceptions, very high-level dynamic data types, dynamic typing and the ability to dynamically generate and execute code. There are interfaces to many system calls and libraries (For example, XML tools). Python is also usable as an extension language for applications that need a programmable interface. The Python implementation is portable: it runs on many brands of UNIX, Linux, on Windows, DOS, OS/2, Mac, and Amiga etc. [24]

Python uses significant white space, which avoid having minor stylistic differences i.e. location of parentheses. Therefore different programmers could work easier due to visual consistency. Technicians who are maintaining code will deal with actual algorithm of the code almost without spending any time to figure out the start and end of class, methods, and loops. These advantages result in lower expense for code maintenance [25].

4.3 Web-based Design Aids

Different web technologies that would be useful in development of design aids, their advantages as well as their disadvantages are discussed.

4.3.1 CGI (Common Gateway Interface)

While CGI technology does not directly help in maintenance of design aids, its use also allows other technologies to play their role. It is a tool to allow web servers to use external programs for computation.

The CGI is a standard for interfacing external applications with information servers, such as HTTP or Web servers. A plain HTML document that the Web daemon retrieves is static, which means it exists in a constant state: a text file that doesn't change, for example. A CGI program, on the other hand, is executed with every access, so that it can output dynamic information.

For example, consider hooking a database to the World Wide Web, to allow users to query it. Basically, a CGI program is needed that the Web daemon will execute to transmit information to the database engine, and receive the results back again and display them to the client.

The database example is a simple idea, but most of the time rather difficult to implement. There is really no limit as to what is hooked up to the Web. The only concern is the fast process of the CGI program. Otherwise, the user has to wait for browser to show the result.

A CGI program can be written in any language that allows it to be executed on the system, such as (C/C++, Fortran, PERL, TCL, Any Unix shell, Visual Basic, AppleScript and Python) [26].

4.3.2 Server feasibility

A server based relational database is used to provide the properties of Canadian and American steel shapes. Using SQL for retrieval, shapes are accessed and changes in database would not affect the structure of the design aid program.

4.3.3 Running on all platforms

Since an electronic handbook may be used on different OS platforms (Windows, Unix, Linux) and on different computers (Mac, PC), it is important to find a method to deal with all of these differences. Web-based design aids can run on any platform and only require one version.

4.3.4 Installation and upgrading

The software is designed to be web-based compatible. This means that it would be easy to incorporate it into a web-based version of the software. This is advantageous in terms of distribution. It should be noted that fast distribution is crucial. After applying the modifications of the design aid, the benefit of the usage of the software depends on its fast distribution. However, providing Standards on CD-ROMs has significantly contributed to the fast production of the Handbook, there

is no guarantee that the user will have the most updated version of the Handbook, even if it is provided free of charge. With web-based Hyper-handbook, a technician can be directed to install the program on the server and then all the users would be automatically updated with new Hyper-handbook as it is readied. The user will always be able to use a previous Hyper-handbook if it is needed. Furthermore installation and upgrading of a Hyper-handbook will not require a technician's help desk and other services, which minimizes the cost of maintenance of programs.

4.3.5 XML

The advantage of using hypertext for any design aid is diminished when it comes to maintenance. A hypertext is very time consuming to debug. XML is what brings back the advantage to the hypertext providing design aids maintenance help. XML does not replace HTML; rather, it is a complementary format.

Features of XML make it ideal for code maintenance. XML has cross-platform portability due to its text-based format. XML's structure is flexible and extending it is so easy.

XML is able to publish data on the Web. XML allows developers to easily describe and deliver rich, structured data from any application in a standard, consistent way.

4.3.5.1 Python and XML

Python supports XML format. Software written in Python generates the desired values and then an XML module presents these values in a table format, which has a convenient hypertext form.

4.3.6 CVS (Concurrent Versions System)

CVS is the abbreviation of Concurrent Versions System that is designed to let individual developers work on the same project.

Concurrent Versions System uses a master repository of source code. CVS saves a complete copy of all files and directories under version control. Every individual works on his own local copies by overwriting the files and periodically commits their changes back to the master repository. CVS manages concurrent editing of source files by recording logs such as time and owner and a brief description of change. CVS also tracks conflicts between changes. The CVS history of the source files can also retrieve old versions. CVS resolves and prevents conflict when two different people modify the same file.

The developers could be a distributed team as well. A client-server access method lets developers access the latest program code from anywhere there is an Internet connection.

CVS technology in this research is used to avoid transfer of files via email between developer and supervisor. CVS also helped in keeping track of the changes and making comparison between them. A CVS tool automatically gets and shows the lines that are added, changed or deleted.

4.4 Pydoc Module

Pydoc generates Python program documentation in HTML or text. Pydoc extracts all the comments that are in the source code and categorizes them as to Class, Function (method) and Data. Comments should be written in specific format within the source code. For example, a method description should be placed after the name and attribute line of the method and started with a `"""` (three quotes) symbol and ended with the same symbol.

Documenting software by Pydoc has the following advantages:

- 1) Documentation is placed in the source code, which facilitates changes that are made to the program later.
- 2) There is not any need for writing documentation other than what is in within the source code.
- 3) Any future changes in the program or documentation do not require rewriting documentation.

4.5 Implementation of Technology

There are different ways of writing a program. To ease the maintenance of programs we should minimize the human intervention steps - extracting formula, ordering parameters and making a link between different parts of the program -in other words whatever the developer would input as human logic. Automating by the substitution of these steps with a computerized process contributes to the maintenance of design aids significantly. The following sections describe necessary considerations in a program to achieve the goal.

4.5.1 Variable classification

As an example, consider an application program to assist in the design of structural steel elements. The design of each structural element requires data derived from a variety of sources. It seems reasonable to differentiate data quantities as coming from one of the following three categories:

1. Those values such as cross-sectional dimensions and lengths. The geometry of the element.
2. Material properties such as yield strength and Young's modulus.
3. Applied loads and internal forces and reactions.

Differentiating between the three categories has advantages as the designer changes data in one category at a time. For example, the design aid should handle different load conditions on a member and generate appropriate results for each one.

Also, maintenance of the program is easier where changes need to be made to isolated categories. The result of isolated categories is less interaction between variables, which will simplify maintenance of programs.

4.5.2 Class and methods and clauses

Due to the independent nature of classes in Object-Oriented Programming, as discussed previously, the design aid software is designed so that each class represents and corresponds with a clause of the Standard. This idea contributes greatly to the ease of modification as changes in any clauses of the Standard are dealt with in a separate class. Since classes are constructed as an independent object, any changes in them do not affect other parts of the program.

4.5.3 Text-based programming

The strategy to make the program source code similar in structure to the text of the Standard and its advantages are discussed in this section.

One of the advantages of having the program source code similar to the text of the Standard is the easy maintainability of the design aids.

The exact same formula as it is used in the text of the Standard is also used in the program source code. For example, in the `mr13p6` method in the `S16p1M78()` class, `Mu` and `Mr` are introduced the same as they appear in the Standard. `Mu` is defined in one line and then `Mr` is defined in another line using the `Mu` definition. In another word, no manual action such as substitution of `Mu` in `Mr` is done.

```
Mu = (pi/omegaValue/aMember.UnbracedLength)*sqrt( aShape.E* \
    aShape.Iy*aShape.G*aShape.J + \
    Shape.Iy*aShape.Cw*(pi*aShape.E \
    /aMember.UnbracedLength)**2.)
```

and the formula for `Mr`:

```
Mr = Phi * Mu
```

Simple Numerical methods are used to get the parameters of the equation, instead of solving it manually. For example, to get `Lu` from the `Mr` equation in the `DesignParameters` class, the `MaxX4MaxY` method, which is a numerical method similar to bisection search for root finding, is used from the `Utils` class.

4.5.4 Data Orientated Style

The programmer can generate a table in a few different ways. One method is to categorize the table content by the logic that they represent. In this method the programmer considers the relationship between cells contents and finds a pattern among them. The only advantage of this method is having short program codes. The main disadvantage is the possibility of violating this pattern when introducing a new change in the Standard. The other method to generate tables that we refer to as Data Orientated

Style (DOS) in this thesis is to use row titles as class names and column titles as arguments. The main advantage of DOS is that the program is in fact like a spreadsheet that is easily editable. For example, in the G4021 method in the SteelProperties class, which corresponds to the mechanical steel properties table in the Handbook, the initialization is as follow:

```
def __init__( self, name, maxt, Fyt1, Fyt2, Fyt3, maxGrp, Fyg3and4, Fyg5,
maxwt, Fyhss,  $F_y=350.0$ ,  $F_u=450.0$ ):
```

In the mechanical steel properties table name, maxt, Fyt1, Fyt2, Fyt3, maxGrp, Fyg3and4, Fyg5, maxwt, Fyhss, $F_y=350.0$, $F_u=450.0$ are titles of the table. Now for 300W material properties we just have to enter a row of table in class format such as:

```
class G4021_300W( G4021 ):
    """Returns a row of table with 300W grade character.
    Default calculations just return nominal values of G4021 by
    inheritance."""

    def __init__( self ):
        G4021.__init__(self, name="CSA G40.21 300W", maxt=100.0,
            Fyt1=300.0, Fyt2=290.0, Fyt3=280.0,maxGrp=3,
            Fyg3and4=290., Fyg5=None, maxwt=16.0, Fyhss=300.,
             $F_y=300.0$ ,  $F_u=450.0$ )
```

The G4021_300W class initializes the values as if we are entering a row of data to the table.

4.5.5 Dictionary

The clauses of Standard usually have more than one parameter that a user would like to calculate, and would do so by running a correspondent class. For example, a

designer uses clause 13.6 for calculating Mr as well as omega2. Running the corresponding method, should calculate both Mr and omega2 parameters for the user.

A dictionary in Python language is a list of paired variables; there is a value that is assigned to each variable. A dictionary is a useful tool for the above-mentioned problem. By using a dictionary all parameters would be calculated and gathered in one place. The user could calculate both variables by running the clause13p6 method that returns values of Mr and omega2 parameters in the dictionary.

If a clause is subjected to changes, there are parameters that should be either omitted or added. Using dictionaries make classes independent of variable names or parameters. For example, if any other parameter is needed to be calculated in the clause13p6 method in the future, the parameter could be calculated and stored in the returned dictionary. Any method that uses the clause13p6 method would remain unchanged, as the output of the clause13p6 method is still the same as long as the name of the dictionary is not changed and the order of the data is not important in the dictionary, like all dictionaries in Python.

4.5.6 Query

A class, which plays the role of a query, would call and run the class and bring back a specific parameter from a method that returns a dictionary such as the clause13p6 method. For example, a query method “calculatorFor” in the S16p1M78() class is used to get Mr by the following statements:

```

anInstance_standard = S16p1M78()
queryMr = anInstance_standard.calculatorFor('Mr')
queryMr(self.aMember, aShape)['Mr']

```

In the first line an instance of the standard is created. In the second line a query (for Mr) is created by calling a method of the standard instance. In the last line Mr is called from a dictionary that is returned by the query method.

In this way a method such as the clause13p6 method could be called automatically by a query method. This means any future changes to the structure of the S16p1M78() class will be handled in the S16p1M78() class itself. In this way other design aids that are using the S16p1M78() class, such as BST class, do not get affected by that change.

The Query idea is extremely useful to accommodate changes where the structure of clauses is changed such as in the latest revision of S16.1 [2].

4.5.7 Data identifier

Shape properties, such as cross-sectional and moment of inertia, are stored in a database called Structural Shape Table (SST). This database is used to provide shape properties for the software. A strategy that is used to get data from the database makes a significant contribution to ease of modification of the software. That strategy is about making the order of columns immaterial. Each row of data is converted to the dictionary

indexed by a column name (such as “A” and “Ixx”). Dictionaries in Python are structured so that calling a parameter in the dictionary returns its value regardless of their index. Therefore numeric data is identified with the column name.

Whenever there are changes in the database there is rarely need to modify the program. Hence, the program uses a meaningful name to access specific data not an index that is subjected to changes. For example, a line of program, `mr13p6` method in class `S16p1M78`, looks like this:

```
Mu = (pi/omegaValue/aMember.UnbracedLength)*sqrt( \
    aShape.E*aShape.Iy*aShape.G*aShape.J + \
    aShape.Iy*aShape.Cw*(pi*aShape.E\
    /aMember.UnbracedLength)**2.)
```

5 OVERVIEW OF SOFTWARE ARCHITECHTURE

5.1 Introduction

Programs have been developed to generate Beam Selection Tables, Beam Load Tables and Class of Sections in bending. The software architecture for the computerized design aid is based on the technologies introduced in former chapters as well as ideas that are implemented in the literature review. Moreover, unique and innovative methods of programming are implemented to accommodate easier modification of the software.

In this chapter different parts of the software, as well as the interaction between them are discussed and justified. The following sections describe the programs and how they contribute to achieving easier maintenance of the design aids.

5.2 Beam Selection Tables

5.2.1 Module structures

The software that generates Beam Selection Tables has 6 modules. All the modules are described and their interactions are discussed in the following sections.

5.2.1.1 S16p1M78 module

This module acts as a database of Standard 16.1-M78. Clauses of standard are the name of classes of the module. Methods of class support formulas of the corresponding clauses of Standard.

5.2.1.2 SST module

The SST module provides high-level functions for accessing the Structural Shape Table (SST) Database. SST contains the Properties and Dimensions data for Structural Steel Shapes as given in Part six of the Handbook.

5.2.1.3 IShape module

An IShape is one structural section from the shape tables (i.e., one complete row from the table). The individual properties are stored as attributes with the proper name. Module IShape provides all the attributes of an IShape from a Structural Shape Table (SST) Database.

5.2.1.4 MemberSegment module

This module specifies loads and geometry (length, unbraced length, moment, etc) for a steel member segment.

The sign convention used in the beam section is illustrated in the next page figures:

Figure 5.1. Positive directions of the loading P and deflection y

Figure 5.2. Positive direction of the bending moment M

Figure 5.3. Positive direction of the shear force V

Notice that unlike some publications on this subject, the loading term $P(x)$ is positive in the direction of the y -axis. Please refer to the Euler-Bernoulli beam theory for more details [27].

5.2.1.5 SteelProperties module

The SteelProperties module corresponds to the mechanical steel properties table in the Handbook. This module has all the steel properties, such as F_y and E and can assign any steel property to steel shapes. Note that numerical values of F_y depend on the grade of steel, type of section, and thickness of plate.

5.2.1.6 Utils module

This module provides functions for numerical methods calculations. Most of the mathematical methods are provided in standard Python libraries such as “math”. Methods that are used specifically for the software and are not provided in the standard Python libraries are included in Utils module.

5.2.1.7 DesignParameters module

There are a few parameters that are used for design purpose but they do not appear in the text of the standard. One example is the maximum unsupported beam length (L_u). This module calculates those parameters by accessing formulas specified in the Standard.

5.2.2 Logical Interaction between modules

The module interactions are shown using UML diagrams. UML is a software notation standard that is used in software development to show either structure or behavior of different parts of software and how they interact with each other.

Package and Class diagrams illustrate the structure of the software and the Sequence diagrams illustrate the behavior of the software.

5.2.2.1 Software packages

The software structure consists of three parts: Standard, Components and Interfaces. The UML package diagram organizes the elements of a system into related groups to minimize dependencies among them. The UML package diagram for the software is shown in Figure 5.4.

Figure 5.4. Package diagram

The system is defined as groups of classes that comprise the software. In a UML package diagram, tabbed folders with the name of the package on the tab represents a system. Each box inside the system is a part of the system with different characteristic that makes them distinguished from other parts. These boxes would be in different

modules later on. Arrows show dependency between systems for example arrow's head is toward the system that imports the module at the other end of the arrow system.

The Standard package acts as a database. Any instance of the Standard package has methods that are in correspondence with clauses of standard S16.1 and is able to calculate all the formulas of the standard S16.1. In other words clauses of standard S16.1 could be extracted from the Standard package instances. Instances of the standard are used to get formulas in other part of software. For example, clause 13.6 of the S16.1 Standard has a formula to calculate Mr. Therefore Standard package S16.1 has a class "clause13p6" representing clause13.6. That class has a method that calculates Mr.

The "Components" package holds classes that provide "geometry and load conditions", "shape properties" and "material properties" specifications of the member to be designed. The "geometry and load conditions", "shape properties" and "material properties" are objects of different nature. Therefore, they should be instances of three different classes. By categorizing "geometry and load conditions", "shape properties" and "material properties" objects as independent classes, it is possible to keep changes in the Standard independent from changes in the member properties and vice versa.

The main part of the program, that controls the programs, gets an instance of Standard and calls Standard methods with specific "Components" to get the values that are needed for the design aid. For example, the main part of the program gets an instance

of S16.1 and calls clause13p6 class providing “Components” such as material properties, geometry and load condition, and shape properties as class arguments.

The result will be sent to an Interface that will put together all the design aid variables in a table that could be published as a web page for the user. It is possible that user inputs data in that interface if it is interactive. The user could customize the design aid by changing the table headers. In the current version of the software no interface is developed due to the fact that interface design was beyond the scope of the research, but the possibility of an interactive interface is foreseen.

5.2.2.2 Software classes interaction

Classes are defined so that each class plays an independent role allowing for the possibility of easy interaction with other classes. OOP features such as inheritance are used.

Figure 5.5. Class diagram

In class diagrams a box represents a class. The text inside shows the class name. For example, G4021 is the name of a class, representing all G40.21 steels.

Relationships between the classes are shown with lines. The line with a triangle at its head represents inheritance, and the line with a diamond at its head represents a strong relationship such as importing relationship. For example, Beam Selection Tables uses a component that is an instance of class G4021_300W and G4021 class inherits from Steel Base class.

5.2.2.3 Software sequence

A sequence diagram shows the series of sequences that take place, the classes that are executed, in what order they are executed and how the classes interact with each other. The sequence diagram object interactions are arranged on a horizontal time axis where time is represented by a dashed line. The main sequence that leads to the results of the design aids which incorporates all classes of the software is shown in Figure 5.6.

Labels of the process steps in the sequence diagram are in the format of “Object: Class” and the boxes drawn over the dashed lines represent the “lifetime” of that object. Lifetime is the duration of the corresponding method’s instance existence. For example, IShape was called by Beam Selection Table and got the attributes of the shape and returned those attributes and would not exist beyond this action.

In the sequence diagram there are three kinds of objects: entity, control, and boundary. The control object such as the Beam Selection Tables class controls the inter-object interactions. Objects of this type instantiate activities and are in charge of conditions that should be met.

Entity objects such as the SST (Steel Shape Tables) and G4021_300W are the basic objects and usually are stored in a database. Interaction between entity objects is conducted via control objects such as the BST in this software. Entity objects do their task independently.

Boundary objects such as the XML interface define the interface with users. The user could use these boundary objects to access different classes of the software.

Figure 5.6. Sequence diagram

5.3 Beam Load Tables

5.3.1 Module structures

The modules that BLT (Beam Load Table) uses are S16pM78, SST, IShape, MemberSegment and SteelProperties, which are the same as the ones that Beam Selection Tables uses. The Beam Load Table module uses the following formulations:

The maximum allowable uniform distributed load on a section is the minimum of “ $8000 \cdot M_r / l$ ” and “ $2 \cdot V_r$ ”. The corresponding format in the Python language is as follows:

```
UDFL1 = round (((8000.*Mr)/(anUnbraced_length)), 0)
#UDFL stands for Uniformly Distributed Factored Loads
UDFL2 = round ((2.*Vr), 0)
UDFL = min(UDFL1, UDFL2)
```

These formulations are different from the formulation used in the BST module. The above calculations were not needed in the BST module.

5.3.2 Logical interaction between modules

BLT is a controller module. A controller module is defined in previous sections after introducing the sequence diagram. BLT module gets geometry, load condition, and material as three instances of different classes. Then using the above-mentioned formulation it outputs the result. So BLT module’s interaction with other modules is the same as the interaction of Beam Selection Table with other modules. Package, Class and the Sequence diagrams are therefore very similar to those used for the BST module and are not shown here.

5.4 Class of Sections in Bending

5.4.1 Module structures

The modules that CST (Class of Section in Bending Table) uses are S16pM78, SST, IShape, MemberSegment and SteelProperties, which are the same as the ones that Beam Selection Table and Beam Load Table use. The Class of Section Table module uses formulations that are needed to calculate the class number. For example, formulations of the clause 11.2 of the S16-M78 Standard are used. That is the only difference of the CST module and BST module in terms of required changes in the source code of module. Therefore CST has exactly the same structure as BST.

5.4.2 Logical Interaction between modules

CST is another controller module. The CST module like the BST (Beam Selection Tables) and BLT (Beam Load Tables) modules gets geometry, load condition, and material as three instances of different classes. For example, CST gets an instance of the S16pM78 module and calls the clause11p2 method, and then the class number is calculated from the formula of clause 11.2 in the S16-M78. Then generated values are stored and presented in a table format. Again the CST module's interaction with other modules is the same as the interaction of the BST and BLT modules with other modules. Package, Class and the Sequence diagrams are the same as those used for BST and BLT modules.

CHAPTER 6

6 SOFTWARE DETAILS AND DOCUMENTATION

6.1 Introduction

This chapter presents the documentation for the developed software as well as their source code. Automatic documentation is done by pydoc that is described in previous chapters from the source code of the software.

6.2 Documentation

6.2.1 Standard S16.1-M87

Python Library Documentation: module S16p1M78

NAME

S16p1M78

FILE

/home/bansari/bansari/projects/evaluation/S16p1M78.py

DESCRIPTION

This module implements "only" those computations specified in S16.1. For example, if S16.1. specifies how to compute M_p , then Module S16.1pM78 does the same.

M_r -the factored moment resistance developed by a member subjected to uniaxial bending moments about a principal axis and where continuous lateral support is

"not" provided to the compression flange.

w2 -coefficient to account for increased moment resistance of a laterally unsupported beam segment when subjected to a moment gradient.(part 2.2 of handbook "Symbols")

w2 is calculated unlike assuming 1 as a default value in page 5-73 of the Handbook.

Phi -Please refer to section 3.1 named "General" and section named 2.1

"Definitions" in the Handbook

CLASSES

S16p1M78

class S16p1M78

Represents computations specified in S16.1. Input Parameters: None.

Output Parameters: None

Methods defined here:

 __init__(self)

 Initializes nothing.

 calculatorFor(self, query)

 Returns the function name which is called to generate the value of 'query'. 'query' could be either 'Vr' or 'Mr'.

 clause11p2(self, member, shape, matl=<G4021_300W: [Steel CSA G40.21 300W]>)

 Returns the class number of the section.

 'member' specifies loads and geometry (length, unbraced length, moment, etc.). 'shape' contains cross-sectional properties of the cross-section. 'matl' is material properties.

 Default value is set to C300W.

 clause13p5(self, member, shape, matl=<G4021_300W: [Steel CSA G40.21 300W]>, Phi=0.900000000000000002)

 Returns Mr the factored moment resistance developed by a member subjected to uniaxial bending moments about a principal axis and where continuous lateral support is provided to the compressive flange for class number 1, 2 and 3 sections.

 'member' specifies loads and geometry (length, unbraced length, moment, etc.). 'shape' contains cross-sectional properties of the cross-section. 'matl' is material properties.

 Default value is set to C300W. 'Phi' is the resistance factor.

 Default is set to .9

clause13p6(self, member, shape, matl=<G4021_300W: [Steel CSA G40.21 300W]>)

This method (as will all 'clause' methods) returns a dictionary of *all* the values computed by this clause. For example, values for Mr, Mu, w2, etc. Return values are for one case only, i.e., for one Lu, one cross-section, etc. It is up to calling programs to call this method as many times as necessary to generate a whole row of values, for example. 'member' specifies loads and geometry (length, unbraced length, moment, etc.). 'shape' contains cross-sectional properties of the cross-section. 'matl' is material properties. Default value is set to C300W.

cr13p3(self, member, shape, matl=<G4021_300W: [Steel CSA G40.21 300W]>, Phi=0.90000000000000002, nn=1.3400000000000001)

Returns Cr -the factored compressive resistance of a member or component; acting at the centroid of that part of the steel area in compression for class number 1,2 and 3 sections. 'member' specifies loads and geometry (length, unbraced length, moment, etc.). 'shape' contains cross-sectional properties of the cross-section. 'matl' is material properties. Default value is set to C300W. 'Phi' is the resistance factor. Default is set to .9, 'nn' is a parameter for compressive resistance. Default is set to 1.34.

vr13p4p1p1(self, member, shape, matl=<G4021_300W: [Steel CSA G40.21 300W]>, Phi=0.90000000000000002, a=0.0)

Returns 'Vr' the factored shear resistance developed by the web of flexural member., for elastic analysis - Sec. 13.4.1.1. 'member' specifies loads and geometry (length, unbraced length, moment, etc.). 'shape' contains cross-sectional properties of the cross-section. 'matl' is material properties. Default value is set to C300W. 'Phi' is the resistance factor. Default is set to .9, 'a' is the distance between stiffeners. Default is set to 0 meaning there is no stiffeners.

Data and non-method functions defined here:

__doc__ = 'Represents computations specified in S16.1. Input Parameter...
str(object) -> string

Return a nice string representation of the object.
If the argument is a string, the return value is the same object.

__module__ = 'S16p1M78'

| str(object) -> string

| Return a nice string representation of the object.

| If the argument is a string, the return value is the same object.

FUNCTIONS

sqrt(...)
sqrt(x)

Return the square root of x.

DATA

C300W = <G4021_300W: [Steel CSA G40.21 300W]>
__file__ = './S16p1M78.pyc'
__name__ = 'S16p1M78'

6.2.2 Beam Selection Tables

Python Library Documentation: module BeamSelectionTable

NAME

BeamSelectionTable

FILE

/home/bansari/bansari/projects/evaluation/BeamSelectionTable.py

DESCRIPTION

This module includes BST class that generates Beam Selection Tables. Beam Selection Tables is a design aid in the Handbook of Steel Construction.

CLASSES

BST

```
class BST
| Generates values for Beam Selection Tables cell exactly as they appear
| in the third edition of the Handbook of Steel Construction. Input
| Parameters: None. Output Parameters: None
|
| Methods defined here:
|
| __init__(self, member, shape_directory, UnbracedLengths, matl=<G4021_300W:
[Steel CSA G40.21 300W]>)
|   Initializes row and column titles of the beam selection table.
|   'member' specifies loads and geometry (length, unbraced length,
|   moment, etc.). 'shape_directory' is a list of cross-sections.
|   UnbracedLengths is list of unbraced lengths that BST would generate
```

values for 'matl' is material properties. Default value is set to C300W.

content(self, anInstance__standard=<S16p1M78.S16p1M78 instance>)
Returns column contents of the beam selection table.
'anInstance__standard' is an instance of standard with S16p1M78() as default.

header(self, titles=['Desig. '])
Returns column titles of the beam selection table.
titles is a list of titles. Default is set to a list that has Desig. only

Data and non-method functions defined here:

__doc__ = 'Generates values for Beam Selection Tables cell ...ut \n ...
str(object) -> string

Return a nice string representation of the object.
If the argument is a string, the return value is the same object.

__module__ = 'BeamSelectionTable'
str(object) -> string

Return a nice string representation of the object.
If the argument is a string, the return value is the same object.

DATA

C300W = <G4021_300W: [Steel CSA G40.21 300W]>
__file__ = './BeamSelectionTable.pyc'
__name__ = 'BeamSelectionTable'

6.2.3 Beam Load Tables

Python Library Documentation: module BeamLoadTable

NAME

BeamLoadTable

FILE

/home/bansari/bansari/projects/evaluation/BeamLoadTable.py

DESCRIPTION

This module includes BLT class that generates Beam Load Tables. Beam Selection Tables is a design aid in the Handbook of Steel Construction.

CLASSES

BLT

```
class BLT
    Generates values for Beam Load Tables cell exactly as they appear
    in the third edition of the Handbook of Steel Construction. Input
    Parameters: None. Output Parameters: None

    Methods defined here:

    __init__(self, member, shape_directory, UnbracedLengths, matl=<G4021_300W:
[Steel CSA G40.21 300W]>)
        Initializes row and column titles of the Beam Load Table.
        'member' specifies loads and geometry (length, unbraced length,
        moment, etc.). 'shape_directory' is a list of cross-sections.
        UnbracedLengths is list of unbraced lengths that BLT would generate
        values for. 'matl' is material properties. Default value is set to
        C300W.

    content(self, anInstance_standard=<S16p1M78.S16p1M78 instance>)
        Returns column contents of the beam selection table.
        'anInstance_standard' is an instance of standard with S16p1M78() as
        default.

    header(self, titles=['Desig. '])
        Returns column titles of the beam selection table.
        titles is a list of titles. Default is set to a list that has
        Desig. only

-----
    Data and non-method functions defined here:

    __doc__ = 'Generates values for Beam Load Tables cell exact...ut \n ...
    str(object) -> string

    Return a nice string representation of the object.
    If the argument is a string, the return value is the same object.

    __module__ = 'BeamLoadTable'
    str(object) -> string

    Return a nice string representation of the object.
    If the argument is a string, the return value is the same object.
```

DATA

```
C300W = <G4021_300W: [Steel CSA G40.21 300W]>
__file__ = './BeamLoadTable.pyc'
__name__ = 'BeamLoadTable'
```

6.2.4 Class of Sections in bending

Python Library Documentation: module ClassSectionTable

NAME

ClassSectionTable

FILE

/home/bansari/bansari/projects/evaluation/ClassSectionTable.py

DESCRIPTION

This module includes CST class that generates Class of Section in Bending Tables. Class of Section in Bending is a design aid in the Handbook of Steel Construction.

CLASSES

CST

class CST

Generates values for Class of Section in Bending Tables cell exactly as they appear in the third edition of the Handbook of Steel Construction.
Input Parameters: None. Output Parameters: None

Methods defined here:

`__init__(self, member, shape_directory, matl=<G4021_300W: [Steel CSA G40.21 300W]>)`

Initializes row and column titles of the beam selection table.
'member' specifies loads and geometry (length, unbraced length, moment, etc.). 'shape_directory' is a list of cross-sections.
'matl' is material properties. Default value is set to C300W.

`content(self, anInstance_standard=<S16p1M78.S16p1M78 instance>)`

Returns column contents of the beam selection table.
'anInstance_standard' is an instance of standard with S16p1M78() as default.

`header(self, titles=['Desig. '])`

Returns column titles of the beam selection table.
titles is a list of titles. Default is set to a list that has Desig. only

Data and non-method functions defined here:

`__doc__` = 'Generates values for Class of Section in Bending... Input ...
`str(object)` -> string

Return a nice string representation of the object.
 If the argument is a string, the return value is the same object.

`__module__` = 'ClassSectionTable'
`str(object)` -> string

Return a nice string representation of the object.
 If the argument is a string, the return value is the same object.

DATA

`A36` = <A36: [Steel ASTM A36]>
`C300W` = <G4021_300W: [Steel CSA G40.21 300W]>
`C350W` = <G4021_350W: [Steel CSA G40.21 350W]>
`__file__` = './ClassSectionTable.pyc'
`__name__` = 'ClassSectionTable'

6.2.5 Design Parameters

Python Library Documentation: module DesignParameters

NAME

DesignParameters

FILE

/home/bansari/bansari/projects/evaluation/DesignParameters.py

DESCRIPTION

This module computes design parameters that is not driven from the formulas specified in the Standard as they appear. This module is based on S16p1M78.

CLASSES

DesignParameters

class DesignParameters

| This class returns design parameters like Lu. Input Parameters: None.
 | Output Parameters: None

| Methods defined here:

```

    Lu(self, member, shape)
        Compute max unsupported beam length for which Mr is applicable (mm)
        . Mr is factored moment resistance for laterally supported member
        (kN.m).

    __init__(self, member, shape)
        Initializes a member and a shape.

-----
Data and non-method functions defined here:

    __doc__ = 'This class returns design parameters like Lu. Input Paramet...
    str(object) -> string

    Return a nice string representation of the object.
    If the argument is a string, the return value is the same object.

    __module__ = 'DesignParameters'
    str(object) -> string

    Return a nice string representation of the object.
    If the argument is a string, the return value is the same object.

```

DATA

```

__file__ = './DesignParameters.pyc'
__name__ = 'DesignParameters'

```

6.2.6 Member Segment

Python Library Documentation: module MemberSegment

NAME

MemberSegment

FILE

/home/bansari/bansari/projects/HHBlib/MemberSegment.py

DESCRIPTION

This module specifies loads and geometry (length, unbraced length, moment, etc) for a steel member segment.

***** Sign Convention *****

The sign convention used in the beam section is defined in the following illustrations. (Please refer to Figure)

Notice that unlike some publications on this subject, the loading term $P(x)$ points to the same direction as y -axis.

Please refer to the Euler-Bernoulli beam theory for more details.

CLASSES

MemberSegment

class MemberSegment

Represents a segment of a member. Each segment of a member is identified by a certain geometry condition or load condition. Input Parameters: None.
Output Parameters: None

Methods defined here:

`__init__(self, Length=0.0, UnbracedLength=6000.0, Mend1=0, Mend2=0.0, Mmax=0.0, SpEnd1='false', SpEnd2='false', Cf=0.0)`

Initializes segment specifications (loads and geometry), Length is member's length. Default is set to 0.0, UnbracedLength is the length of segment which defined by it's braced barriers. Default is set to 6000., Mend1 is the moment at end 1 of segment. Default is set to 0.0, Mend2 is the moment at end 2 of segment. Default is set to 0.0, Mmax is the max moment between end 1 to 2, but only if greater than end moments. Default is set to 0.0, SpEnd1 and SpEnd2 are support booleans at end 1 and 2 respectively, which are true if comp flange are supported at that end. Default is set to 'false'. Cf is the axial force that acts on the member. Default is set to 0.0, meaning that there is no axial force.

Data and non-method functions defined here:

`__doc__` = 'Represents a segment of a member. Each segment o...ut Param...
`str(object)` -> string

Return a nice string representation of the object.
If the argument is a string, the return value is the same object.

`__module__` = 'MemberSegment'
`str(object)` -> string

Return a nice string representation of the object.
If the argument is a string, the return value is the same object.

DATA

`__file__` = './MemberSegment.pyc'
`__name__` = 'MemberSegment'

6.2.7 Steel Properties

Python Library Documentation: module SteelProperties

NAME

SteelProperties

FILE

/home/bansari/bansari/projects/evaluation/SteelProperties.py

DESCRIPTION

SteelProperties module set default materials for all section types, and will allow an application program to set different default. All strength calculations use this module to get F_y , F_u , E and G unless these have been specified explicitly when the method is called.

SteelProperties module is a computerized version of steel properties table in the Handbook that assigns all steel properties. These properties are exactly same as what is in Table 6-1 to 6-3 in 3rd Edition of the Handbook. ASTM properties is also added due to the fact that other design aids such Class of Section in bending is using it.

The SteelProperties module defines the following classes:

SteelBase

Class SteelBase assigns nominal F_y , F_u , E and G . Methods exist to retrieve nominal F_y , F_u , E and G one by one or whole as a dictionary for a shape by its designation.

G4021

Class G4021 inherits SteelBase and it is a super Class too.

G4021_300W

Class G4021_300W instances contain the steel properties for steel of grade 300W. It is an inheritant of G4021.

G4021_350W

Class G4021_300W instances contain the steel properties for steel of grade 350W. It is an inheritant of G4021.

ASTM

Class ASTM inherits SteelBase and it is a super Class too.

A36

Class A36 instances contain the steel properties for steel of grade A36. It is an inheritant of ASTM.

GradeTable

Class GradeTable provides get and set methods. It can be called by its abbreviation name "Grade" too.

GroupExceedsMaximum

This exception is raised when a shape specified group is not found in the mechanical steel properties table.

ThicknessExceedsMaximum

This exception is raised when a shape specified thickness is greater than Nominal Max thickness in the mechanical steel properties table.

The module has a test program at the end of it that tests the module by assigning different grades to different shapes.

CLASSES

exceptions.Exception

GroupExceedsMaximum

ThicknessExceedsMaximum

GradeTable

SteelBase

ASTM

G4021

G4021_300W

G4021_300WT

G4021_350W

class ASTM(SteelBase)

This class implements the material property calculations for ASTM

Methods defined here:

Fy(self, shape=None)

Returns actual Fy for a specific shape. 'shape' is the steel section designation. Default is set to None which means assigned value will be nominal value defined in SteelBase class

__init__(self, name, Fy1, Fy=350.0, Fu=450.0)

Initializes nominal value of basic steel properties in the order of "table name", Fy1, Fy and Fu. Fy1 is another nominal value assigned for Fy for ASTM steel. Default values are: Fy = 350.0 and Fu = 450.0. "table name", Fy and Fu initialized by inheritance from SteelBase class.

Data and non-method functions defined here:

```
__doc__ = 'This class implements the material property calculations fo...
str(object) -> string
```

Return a nice string representation of the object.
If the argument is a string, the return value is the same object.

```
__module__ = 'SteelProperties'
str(object) -> string
```

Return a nice string representation of the object.
If the argument is a string, the return value is the same object.

Methods inherited from SteelBase:

`E(self, shape=None)`
Returns the E for a specific shape. 'shape' is the steel section designation. Default is set to None which means nominal values would be assigned, nominalE=200000.0

`Fu(self, shape=None)`
Returns the Fu for a specific shape. 'shape' is the steel section designation. Default is set to None which means nominal values would be assigned, Fu=450.0

`G(self, shape=None)`
Returns the G for a specific shape. 'shape' is the steel section designation. Default is set to None which means nominal values would be assigned, nominalG=77E3

```
__repr__(self)  
Produce the string representation of the matl object.
```

```
__str__(self)  
Return a string suitable for printing.
```

`properties(self, shape=None, n=4)`
Returns a tuple of up to 4 properties in the order Fy, Fu, E and G. 'shape' is the steel section designation. Default is set to None which means nominal values would be assigned, Fy=350.0, Fu=450.0, nominalE=200000.0, nominalG=77E3. 'n' is number of sections that should be selected. Default is set to 4

class G4021(SteelBase)

This class implements the material property calculations for CSA G40.21. Input Parameters: None. Output Parameters: None

Methods defined here:

`Fy(self, shape=None)`

Returns actual Fy for a specific shape. 'shape' is the steel section designation. Default is set to None which means nominal values will be assigned, Fy=350.0

`__init__(self, name, maxt, Fyt1, Fyt2, Fyt3, maxGrp, Fyg3and4, Fyg5, maxwt, Fyhss, Fy=350.0, Fu=450.0)`

Initializes column titles of the Steel Properties table. Default values are: Fy=350.0, Fu=450.0

Data and non-method functions defined here:

`__doc__ = 'This class implements the material property calc.... Input ...
str(object) -> string`

Return a nice string representation of the object.
If the argument is a string, the return value is the same object.

`__module__ = 'SteelProperties'`
`str(object) -> string`

Return a nice string representation of the object.
If the argument is a string, the return value is the same object.

Methods inherited from SteelBase:

`E(self, shape=None)`

Returns the E for a specific shape. 'shape' is the steel section designation. Default is set to None which means nominal values would be assigned, nominalE=200000.0

`Fu(self, shape=None)`

Returns the Fu for a specific shape. 'shape' is the steel section designation. Default is set to None which means nominal values would be assigned, Fu=450.0

`G(self, shape=None)`

Returns the G for a specific shape. 'shape' is the steel section designation. Default is set to None which means nominal values would be

assigned, nominalG=77E3

`__repr__(self)`

Produce the string representation of the matl object.

`__str__(self)`

Return a string suitable for printing.

`properties(self, shape=None, n=4)`

Returns a tuple of up to 4 properties in the order Fy, Fu, E and G.

'shape' is the steel section designation. Default is set to None which

means nominal values would be assigned, Fy=350.0, Fu=450.0,

nominalE=200000.0, nominalG=77E3. 'n' is number of sections that should be selected. Default is set to 4

`class G4021_300W(G4021)`

Returns a row of table with 300W grade character.

Default calculations just return nominal values of G4021 by inheritance.

Input Parameters: None. Output Parameters: None

Method resolution order:

G4021_300W

G4021

SteelBase

Methods defined here:

`__init__(self)`

Initializes row values for each column title of the Steel Properties table.

Data and non-method functions defined here:

`__doc__` = 'Returns a row of table with 300W grade character... Input ...
`str(object)` -> string

Return a nice string representation of the object.

If the argument is a string, the return value is the same object.

`__module__` = 'SteelProperties'

`str(object)` -> string

Return a nice string representation of the object.

If the argument is a string, the return value is the same object.

 Methods inherited from G4021:

`Fy(self, shape=None)`

Returns actual Fy for a specific shape. 'shape' is the steel section designation. Default is set to None which means nominal values will be assigned, Fy=350.0

 Methods inherited from SteelBase:

`E(self, shape=None)`

Returns the E for a specific shape. 'shape' is the steel section designation. Default is set to None which means nominal values would be assigned, nominalE=200000.0

`Fu(self, shape=None)`

Returns the Fu for a specific shape. 'shape' is the steel section designation. Default is set to None which means nominal values would be assigned, Fu=450.0

`G(self, shape=None)`

Returns the G for a specific shape. 'shape' is the steel section designation. Default is set to None which means nominal values would be assigned, nominalG=77E3

`__repr__(self)`

Produce the string representation of the matl object.

`__str__(self)`

Return a string suitable for printing.

`properties(self, shape=None, n=4)`

Returns a tuple of up to 4 properties in the order Fy, Fu, E and G. 'shape' is the steel section designation. Default is set to None which means nominal values would be assigned, Fy=350.0, Fu=450.0, nominalE=200000.0, nominalG=77E3. 'n' is number of sections that should be selected. Default is set to 4

`class G4021_300WT(G4021)`

Returns a row of table with 300WT grade character. Default calculations just return nominal values of G4021 by inheritance. Input Parameters: None. Output Parameters: None

Method resolution order:

G4021_300WT

G4021
SteelBase

Methods defined here:

`__init__(self)`
Initializes row values for each column title of the Steel Properties table.

Data and non-method functions defined here:

`__doc__` = 'Returns a row of table with 300WT grade characte...ut Param...
`str(object)` -> string

Return a nice string representation of the object.
If the argument is a string, the return value is the same object.

`__module__` = 'SteelProperties'
`str(object)` -> string

Return a nice string representation of the object.
If the argument is a string, the return value is the same object.

Methods inherited from G4021:

`Fy(self, shape=None)`
Returns actual Fy for a specific shape. 'shape' is the steel section designation. Default is set to None which means nominal values will be assigned, Fy=350.0

Methods inherited from SteelBase:

`E(self, shape=None)`
Returns the E for a specific shape. 'shape' is the steel section designation. Default is set to None which means nominal values would be assigned, nominalE=200000.0

`Fu(self, shape=None)`
Returns the Fu for a specific shape. 'shape' is the steel section designation. Default is set to None which means nominal values would be assigned, Fu=450.0

`G(self, shape=None)`

Returns the G for a specific shape. 'shape' is the steel section designation. Default is set to None which means nominal values would be assigned, nominalG=77E3

`__repr__(self)`

Produce the string representation of the matl object.

`__str__(self)`

Return a string suitable for printing.

`properties(self, shape=None, n=4)`

Returns a tuple of up to 4 properties in the order Fy, Fu, E and G. 'shape' is the steel section designation. Default is set to None which means nominal values would be assigned, Fy=350.0, Fu=450.0, nominalE=200000.0, nominalG=77E3. 'n' is number of sections that should be selected. Default is set to 4

`class G4021_350W(G4021)`

Returns a row of table with 350W grade character. Default calculations just return nominal values of G4021 by inheritance. Input Parameters: None. Output Parameters: None

Method resolution order:

G4021_350W

G4021

SteelBase

Methods defined here:

`__init__(self)`

Initializes row values for each column title of the Steel Properties table.

Data and non-method functions defined here:

`__doc__` = 'Returns a row of table with 350W grade character...ut Param...
`str(object)` -> string

Return a nice string representation of the object.
If the argument is a string, the return value is the same object.

`__module__` = 'SteelProperties'
`str(object)` -> string

Return a nice string representation of the object.

If the argument is a string, the return value is the same object.

Methods inherited from G4021:

Fy(self, shape=None)

Returns actual Fy for a specific shape. 'shape' is the steel section designation. Default is set to None which means nominal values will be assigned, Fy=350.0

Methods inherited from SteelBase:

E(self, shape=None)

Returns the E for a specific shape. 'shape' is the steel section designation. Default is set to None which means nominal values would be assigned, nominalE=200000.0

Fu(self, shape=None)

Returns the Fu for a specific shape. 'shape' is the steel section designation. Default is set to None which means nominal values would be assigned, Fu=450.0

G(self, shape=None)

Returns the G for a specific shape. 'shape' is the steel section designation. Default is set to None which means nominal values would be assigned, nominalG=77E3

__repr__(self)

Produce the string representation of the matl object.

__str__(self)

Return a string suitable for printing.

properties(self, shape=None, n=4)

Returns a tuple of up to 4 properties in the order Fy, Fu, E and G. 'shape' is the steel section designation. Default is set to None which means nominal values would be assigned, Fy=350.0, Fu=450.0, nominalE=200000.0, nominalG=77E3. 'n' is number of sections that should be selected. Default is set to 4

class GradeTable

| This class maintains default steel grades for each type of section.

| Input Parameters: None. Output Parameters: None

| Methods defined here:

`__init__(self)`

Initializes an empty dictionary.

`getSteel(self, shpCode)`

Return the default grade of steel for this shape. 'shpCode' is the code that is assigned with each steel section designation.

`setSteel(self, shpCode, steelGrade)`

Sets the default grade of steel to use for shapes with given shape designation code. 'shpCode' is assigned with each steel section designation. 'steelGrade' is the grade of mechanical properties of the steel.

Data and non-method functions defined here:

`__doc__ = 'This class maintains default steel grades for ea... Input ...`
`str(object) -> string`

Return a nice string representation of the object.
If the argument is a string, the return value is the same object.

`__module__ = 'SteelProperties'`
`str(object) -> string`

Return a nice string representation of the object.
If the argument is a string, the return value is the same object.

`class GroupExceedsMaximum(exceptions.Exception)`

Data and non-method functions defined here:

`__doc__ = None`

`__module__ = 'SteelProperties'`
`str(object) -> string`

Return a nice string representation of the object.
If the argument is a string, the return value is the same object.

Methods inherited from exceptions.Exception:

`__getitem__(...)`

`__init__(...)`

`__str__ (...)`

`class SteelBase`

Assigns nominal F_y , F_u , E and G . Methods exist to retrieve nominal F_y , F_u , E and G one by one or whole as a dictionary for a shape by its designation. Input Parameters: None. Output Parameters: None

Methods defined here:

`E(self, shape=None)`

Returns the E for a specific shape. 'shape' is the steel section designation. Default is set to None which means nominal values would be assigned, $\text{nominal}E=200000.0$

`Fu(self, shape=None)`

Returns the F_u for a specific shape. 'shape' is the steel section designation. Default is set to None which means nominal values would be assigned, $F_u=450.0$

`Fy(self, shape=None)`

Returns actual F_y for a specific shape. 'shape' is the steel section designation. Default is set to None which means nominal values would be assigned, $F_y=350.0$

`G(self, shape=None)`

Returns the G for a specific shape. 'shape' is the steel section designation. Default is set to None which means nominal values would be assigned, $\text{nominal}G=77E3$

`__init__(self, name, nominalFy, nominalFu, nominalE=200000.0, nominalG=77000.0)`

Initializes nominal value of basic steel properties in the order of "table name", F_y , F_u , E and G . Default nominal values are: $\text{nominal}E=200000.0$ and $\text{nominal}G=77E3$.

`__repr__(self)`

Produce the string representation of the matl object.

`__str__(self)`

Return a string suitable for printing.

`properties(self, shape=None, n=4)`

Returns a tuple of up to 4 properties in the order F_y , F_u , E and G . 'shape' is the steel section designation. Default is set to None which means nominal values would be assigned, $F_y=350.0$, $F_u=450.0$,

nominalE=200000.0, nominalG=77E3. 'n' is number of sections that should be selected. Default is set to 4

Data and non-method functions defined here:

__doc__ = 'Assigns nominal Fy, Fu, E and G. Methods exist t.... Input ...
str(object) -> string

Return a nice string representation of the object.
If the argument is a string, the return value is the same object.

__module__ = 'SteelProperties'
str(object) -> string

Return a nice string representation of the object.
If the argument is a string, the return value is the same object.

class ThicknessExceedsMaximum(exceptions.Exception)

Data and non-method functions defined here:

__doc__ = None

__module__ = 'SteelProperties'
str(object) -> string

Return a nice string representation of the object.
If the argument is a string, the return value is the same object.

Methods inherited from exceptions.Exception:

__getitem__(...)

__init__(...)

__str__(...)

DATA

A36 = <A36: [Steel ASTM A36]>

C300W = <G4021_300W: [Steel CSA G40.21 300W]>

C300WT = <G4021_300WT: [Steel CSA G40.21 300WT]>

C350W = <G4021_350W: [Steel CSA G40.21 350W]>

Grades = <SteelProperties.GradeTable instance>

Shp_HSS_RCT = 16

Shp_HSS_RND = 17

```

Shp_HSS_SQ = 15
Shp_WRF = 14
Shp_WWF = 5
Shp_WWT = 10
__ID__ = '$Id: SteelProperties.py,v 1.2 2003/04/10 00:02:05 bansari Exp ...
__file__ = './SteelProperties.pyc'
__name__ = 'SteelProperties'

```

6.2.8 XML Table

Python Library Documentation: module XMLtable

NAME

XMLtable - This module provides tools to generate tables in XML format.

FILE

/home/bansari/bansari/projects/evaluation/XMLtable.py

CLASSES

Page

class Page

| Generates a document page (i.e. a table) in XML format.

| Input Parameters: None. Output Parameters: None

| Methods defined here:

| `__init__(self)`

| Initializes document and a table.

| `addRow(self, values, emph=[])`

| Creates a row of a table. Row's CELL are elements of "values" list.

| Attributes of the elements that are in the emph list too get set to
 | ('EMPH', '1'). emph default is an empty list.

| `pprint(self)`

| Prints the document.

 | Data and non-method functions defined here:

| `__doc__` = 'Generates a document page (i.e. a table) in XML ... Input ...
 | `str(object)` -> string

| Return a nice string representation of the object.

| If the argument is a string, the return value is the same object.


```
__module__ = 'XMLtable'
str(object) -> string
```

Return a nice string representation of the object.
If the argument is a string, the return value is the same object.

DATA

```
__file__ = './XMLtable.pyc'
__name__ = 'XMLtable'
implementation =
<xml.dom.html.HTMLDOMImplementation.HTMLDOMImplementa...
```

6.2.9 Utils

Python Library Documentation: module Utils

NAME

Utils

FILE

/home/bansari/bansari/projects/HHBlib/Utils.py

DESCRIPTION

This module provides calculation tools that is needed during the design with Canadian Steel Code.

FUNCTIONS

maxX4maxY(input_function, lowerBound, upperBound,
inputFunctionTolerance=0.01, Xtolerance=0.01)

This function returns X Max corresponding to Max of input_function(x) value, given a function in input_function(X) format. 'input_function' is a function to find the Max X of Max [input_function(x)]. 'lowerBound' is lower bound on X. 'upperBound' is upper bound on X. 'inputFunctionTolerance' is the input_function's tolerance value. Default tolerance value is set to .01 'Xtolerance' is the input_function's X tolerance value. Default tolerance value is set to .01.

DATA

```
__file__ = './Utils.pyc'
__name__ = 'Utils'
```

6.2.10 IShape

Python Library Documentation: module IShape

NAME

IShape

FILE

/home/bansari/bansari/projects/HHBlib/IShape.py

DESCRIPTION

Module IShape provides all attributes of an IShape from a Structural Shape Table (SST) Database.

CLASSES

SST.Shape

IShape

```
class IShape(SST.Shape)
```

```
| An IShape is one structural section from the shape tables
| (i.e., one complete row from the table). The properties are
| stored in attributes of the correct name.
```

```
| Methods defined here:
```

```
| __init__(self, row=[], columns=[], matl=None)
```

```
| Return an instance, with all attributes initialized from a row
| of the SST. 'row' and 'columns' are row and columns of the SST.
| Defaults are empty lists. 'matl' matl is the mechanical properties.
| Default is set to None which means nominal values would be assigned,
| Fy=350.0, Fu=450.0, nominalE=200000.0, nominalG=77
```

```
| set_matl(self, matl=None)
```

```
| Set the material type and material properties of the shape. 'matl' is
| the mechanical properties. Default is set to None which means
| nominal values would be assigned, Fy=350.0, Fu=450.0, nominalE=200000.0,
| nominalG=77
```

```
-----
| Data and non-method functions defined here:
```

```
| __doc__ = 'An IShape is one structural section from the sha...are\n ...
| str(object) -> string
```

```
| Return a nice string representation of the object.
| If the argument is a string, the return value is the same object.
```

```
| __module__ = 'IShape'
```

```
| str(object) -> string
```

Return a nice string representation of the object.
 If the argument is a string, the return value is the same object.

 Methods inherited from SST.Shape:

`__repr__(self)`
 Produce the string representation of the shape object.

`__str__(self)`
 Produce a simple printable representation of the object.
 This is used by the 'print' statement to produce output.

`docs(self)`
 Return the documentation tuple.

DATA

`Grades = <SteelProperties.GradeTable instance>`

`__file__ = './IShape.pyc'`

`__name__ = 'IShape'`

7 EXAMPLES AND EVALUATION

In this chapter the developed software is evaluated in terms of lending itself to changes. Also a few examples are shown to describe functionality of the software.

7.1 Introduction

To evaluate the methodology, design aid software was created for an earlier version of the Handbook and was successfully modified in stages to correspond to several later versions. In particular a small number of design aids from the 3rd edition of the Handbook was adapted to generate design aids for the 4th through 8th editions of the Handbook. This chapter examines the modification of these design aids.

7.2 Testing

Each module of the software is tested individually and the whole software set is also tested as a package.

7.2.1 Module tests

Each module contained its own testing and debugging code that was executed whenever the module was changed. This extensive testing machinery allowed testing of

each module separately from testing the other modules that depended on it. This ensured that a larger program was always built on robust and validated code.

7.2.2 Package tests

The results generated by the design aids software packages were compared with the values displayed in the Handbook. Calculations were also performed manually to confirm the test results.

7.3 Evaluation

Evaluation is made to determine how the software lends itself to the updates required by changing standards and usage. For this purpose, the software is modified to generate design aids exactly as they appear in edition 3rd, 4th, 5th, 6th, 7th and the proposed 8th edition of the Handbook. The proposed 8th edition of the Handbook accommodates all changes that, to the time of writing this thesis, needed to be applied to the design aids.

7.3.1 Source code changes measurement

The strategy used to evaluate the developed design aids are based on the number and type of changes that are made to the software in terms of time, line of codes and number of modules. The programming time that is spent making changes is subject to the skills of the programmer and therefore not easily quantified. The number of lines added, deleted or changed in the source code of software is considered the most tangible measure for ease of change factor.

As an instance the extent of changes needed in the design aids to accommodate changes from edition 4th to 5th in terms of changed lines of software is tabulated in the Table 7.1.

Table 7.1. Required Changes to Obtain Fifth Edition of The Handbook from the Fourth edition

Changes in Beam Selection Tables	Changes in Software		Program Lines		
			Added	Changed	Deleted
Based on CSA standard CAN/CSA-S16.1-M89	Use module S16p1M89	Clause 11.2	-----	5	6
		Clause 13.6	-----	1	-----
		Clause 13.4.1.1	-----	6	4
In properties column V_r , I_x , b and L_u were reordered also J and C_w were omitted	Corresponding change in the BeamSelectionTable module		-----	3	2
Totally in SI unit	Change in database		-----	-----	-----
Part 5 (flexural members), additional tables for WWF beams in CSA G40.21 grade 350W steel.	Change in database		6*	1*	-----
Tables related to mechanical and chemical properties of steel revised according to approved changes for 1991	Corresponding change in the SteelProperties module		-----	16**	3

* There is not any change in hard code of program. The numbers are changed input lines in the Beam Selection Tables module

** Data type changes

Table 7.1 tabulated required changes in the Beam Selection Tables in the 5th edition of The Handbook and the corresponding changes that were required in the software. The name of module that was modified to accommodate the changes is shown under the “software” column. The “Program Lines” column shows how many lines of software source code were added, changed or deleted. The numbers provide a tangible measure to adaptability of the software to desired changes. The numbers are measured by CVS - introduced in a previous chapter.

As Table 7.1 shows each change requires changes in only one module. This fact contributes significantly to an easier modification of software. A common change is “Change in database”, that is almost an automatic process and is not more than add/delete a number of data row in the database.

7.3.2 Scalable modules

All the modules developed are expandable. For example, in the SteelProperties module adding an ASTM class, which includes ASTM material properties, was very straightforward.

7.3.3 Easy to plug modules

Each module is designed so that it has minimum dependency on other modules. Each module represents a different aspect of design. For instance, material is represented by the SteelProperties module and load by the MemberSegment module.

This feature enables design aids to work with different Standards. As the Standard changes from one to another, the design aids modules do not require significant change. The new standard module is substituted in different versions of the Handbook like the one that Table 7.1 illustrates. The only change to the other modules would be change of name if the substituted module’s name were different. More research will enrich standard module in terms of its efficiency. Eventually a database-oriented module is foreseen to replace the presented standard module in this research.

7.4 Summary

The first step in applying changes to the design aids is to identify their origin. Standard, material properties shape properties or load and geometry input variables are the major sources of changes. As it is shown in this chapter each change in the “Standard”, “material properties” or “load and geometry condition” requires changes in few lines of source code in usually just one module. Changes to shape properties are due to change in the database, which is an almost automatic process and is not particularly relevant to this research.

8 SUMMARY AND CONCLUSIONS

In this chapter a summary of the research is provided, and the implementation and the effect of the contributed technology is discussed. Finally, a few streams are introduced for further research.

8.1 Summary

This thesis has introduced the problem of maintenance of the software for the design aids in the Steel Design Handbook and has developed an evaluation methodology to study the effects of changes. Design aid software was developed using one potential technology to improve ease of maintenance. The resulting software was studied to demonstrate that it meets the requirements for easing the maintenance of design aids.

Maintenance of design aids in the Handbook is an ongoing task as the Standard changes, steel properties are changed, and the shape sections are modified. The current computerized format of the Handbook software is proposed to be able to keep up with the growing rate of changes in the standard. The software that is developed in this thesis is

tested and evaluated to rapidly accommodate these changes in the Standard. The software enhances the Handbook's functionality.

The architecture of design aids software has been studied to find an appropriate setting and environment for software development.

One important characteristic for the software was to match its architecture to the categorized structure of steel member's properties. The software components, which are referred to as objects in OOP (Object-Oriented Programming), are representative of each category of properties.

It was discovered during the study that modification of design aids is easier when changes are made to objects. Different technologies and how they should be implemented are studied and the significance of each technology explained with examples. Then the developed design aids are studied in terms of the changes they require to adapt them to different versions of the standard over the past 20 years.

One suitable environment is investigated and used for developing software. The environment includes the Linux operating system, the Python programming language and CVS source code control technology. It was discovered that the most important characteristics of the environment included such features as platform independence, fast tracing of changes, easy syntax and the ability to host software for a potential web based version of the software.

8.2 Conclusion

Programs developed for the Hyper-handbook showed that planning for modification techniques is reasonable and desirable. They also illustrate the implementation of efficient and cost effective technologies to maintain the design aids. It was discovered that software architecture plays a key role in the maintenance of design software. The technologies that are used in the software required a suitable environment for the development and maintenance of design aids. The ability of easier modification to the software was demonstrated by tracking the effect of changes over the past two decades.

The clauses of the design standard all have the same structure. They have a title, a text body, and in most case a few formulae. Therefore techniques introduced can also serve other clauses of the steel design standard. Other standards such as the wood engineering code have the same structure as well. Therefore it is possible to manipulate the techniques proposed in this thesis and use the techniques for other standards too. To take full advantage of the new technologies, however, research is required in each discipline (such as wood engineering) to evaluate the technologies that best suit each standard structure.

8.3 Future Research

Having this technology in place could lead to a multitude of interesting results. One of the most interesting areas might be to develop modules such as a help module into the integrated system for the analysis, design, fabrication and erection of steel structures.

Another area that requires more research is improving modules of the standard. Handcrafted modules of standard clauses were used in the software as a substitute for a database that represents the standard. A standard database could be developed to represent the standard and generate these modules semi-automatically, and that is the subject of a current study [29].

Examples of usage of each of the design aids that are in the Handbook could be presented in the computerized version of the Handbook too, that requires its own investigation (i.e. an example that illustrates how to design a beam using Beam Selection Tables).

The evaluation method that is used in this research could be a catalyst to discuss the maintenance of any software that computer programmers develop. More research in maintenance could contribute to the betterment of programs and lengthen their usage.

REFERENCES

References are listed in IEEE Style [28].

[1] Ontario. Building Branch and Ministry of Housing, *Ontario Regulation 419*, Toronto: Ministry of Housing, 1986.

[2] Canadian Standards Association, *Limit States Design of Steel Structures*: Toronto, ON, various editions.

[3] Canadian Institute of Steel Construction, *Handbook of Steel Construction*, Alliston, ON: Universal Offset Limited, various editions.

[4] S. J. Fenves, "Tabular decision logic for structural design", *ASCE (ST1) Journal of the Structural Division*, pp. 473-490, December 1966.

[5] S. K. Goel and S. J. Fenves, "Computer-aided processing of design specification", *ASCE (ST6) Journal of the Structural Division*, vol. 97, pp.463-479, January 1971.

[6] D. J. Nyman and S. J. Fenves, "Organizational model for design specifications", *ASCE (ST4) Journal of the Structural Division*, vol. 101, pp.697-716, April 1975.

[7] S. J. Fenves, K. Rankin and H. K. Tejuja, "The structure of building specifications," " NBS BSS 90, NBS Washington, DC, 1976.

- [8] S. J. Fenves and R. N. Wright, "The representation and use of design specifications Computer aided engineering problems," *Technical Note 940, NBS, Washington*, 1977.
- [9] J. R. Harris and R. N. Wright, "Organization of Building Standards; Systematic Techniques for scope and arrangement," *NBS BSS 136, NBS Washington, DC*, 1980.
- [10] S. J. Fenves, R. N. Wright, F. I. Stahl and K. A. Reed, "Introduction to SASE: Standard analysis, synthesis and expression," *NBSIR 873513, NBS Washington, DC*, 1987.
- [11] D. R. Rehak and L. A. Lopez, "Computer aided engineering problems and prospect." *Civil Engineering Standards SRS8; Department of Civil Engineering, University of Illinois at Urbana -Champaign*, 1981.
- [12] L. A. Lopez and S. L. Elam, "A prototype knowledge based system for conformance checking and design" *Technical Reports; Department of Civil Engineering University of Illinois at Urbana-Champaign*, 1984.
- [13] S. J. Fenves and J. H. Jr. Garrett, "A knowledge-based standard processor for structural component design," *Journal of Engineering with Computers*, vol. 2, pp.219-238, 1987.

- [14] W. J. Rasdorf and T. E. Wang, "Generic design standards processing in an expert system environment, " *Journal of Computing in Civil Engineering*, vol. 2, pp. 68-87, January 1988.
- [15] N. Yabuki and K. H. Law, "An Object-Logic model for representation and processing of design standards," *Journal of Engineering with Computers*, vol. 9, pp. 133-159, 1993.
- [16] J. H. Jr. Garrett and M. M. Hakim, "A description logic approach for representing engineering design standard," *Journal of Engineering with Computers*, vol. 9, pp.108-124, 1993.
- [17] S. M. Cornick "HyperCode: The building code as a HyperDocument." *Journal of Engineering with Computers*, pp.37-46, 1991.
- [18] S. Malasri, J.C. Olabe and M. A. Olabe, "A prototype of the uniform building code using Hypertext." *ASCE Proceedings of 10th Conference on Electronic Computation*, pp.36-43, 1991.
- [19] N. Yabuki and K. H. Law, "HyperDocument model for design standard documentation," *Journal of Computing in Civil Engineering*, vol. 7, pp.218-237, April 1993.

- [20] Frederick M. Williams, "A design standards document server," M. Eng. Thesis, Carleton University, Ottawa, ON, Canada, 1994.
- [21] L. Frattaroli, "Hyper-handbook of Steel Construction," M. Eng. Project, Carleton University, Ottawa, ON, Canada, 1999.
- [22] Z. Turk and D. Dana, "Tools and models for the electronic delivery of building codes and standards" in *CIB Proceedings Publication 180, Modeling of Buildings through their life cycle*, 1995, pp. 20-30.
- [23] M. H. Ackroyd, S. J. Fenves and W. McGuire, "Computerized LRFD specification," June 1988.
- [24] Python Software Foundation, "What is Python?," 2003, <http://www.python.org/doc/Summary.html>
- [25] C. Jones and F. J. Drake, "Python easy to maintain," in *Python programmers to XML processing*, O'Reilly and Associates, 2002.
- [26] University of Illinois at Urbana - Champaign, "Common Gateway Interface," 2003, <http://hoohoo.ncsa.uiuc.edu/cgi/intro.html>

[27] Engineering fundamentals, "Sign Convention for Beams," 2003,
http://www.efunda.com/formulae/solid_mechanics/beams/sign_convention.cfm

[28] Reference Guide: IEEE Style, University of Illinois at Urbana-Champaign, "THE REFERENCE LIST", April 1998, http://www.ece.uiuc.edu/pubs/ref_guides/ieee.html

[29] John Buck, "Representation and Automatic Translation of Design Standards", M. A. Sc. (In Progress), Carleton University, 2003-

APPENDIX A

Source codes

Standard S16.1-M87

Source code for Standard S16.1-M87 is provided in following pages.

```
#!/usr/bin/env python

"""
This module implements "only" those computations specified in S16.1. For
example, if S16.1. specifies how to compute  $M_p$ , then Module S16.1pM78 does
the same.

 $M_r$  -the factored moment resistance developed by a member subjected to uniaxial
bending moments about a principal axis and where continuous lateral support is
"not" provided to the compression flange.
 $w_2$  -coefficient to account for increased moment resistance of a laterally
unsupported beam segment when subjected to a moment gradient.(part 2.2 of
handbook "Symbols")
 $w_2$  is calculated unlike assuming 1 as a default value in page 5-73 of the
Handbook.
 $\Phi$  -Please refer to section 3.1 named "General" and section named 2.1
"Definitions" in the Handbook"""

#precise import prevents lengthy document extracted by pydoc for unused function
import sys
sys.path.insert( 0, ".." )
from SteelProperties import C300W
from math import sqrt

class S16p1M78:

    """Represents computations specified in S16.1. Input Parameters: None.
    Output Parameters: None"""

    def __init__( self):
        """Initializes nothing."""

    def clause13p6( self, member, shape, matl=C300W):
        """This method (as will all 'clause' methods) returns a
        dictionary of *all* the values computed by this clause.
        For example, values for  $M_r$ ,  $M_u$ ,  $w_2$ , etc. Return values
        are for one case only, i.e., for one  $L_u$ , one cross-section, etc.
```

It is up to calling programs to call this method as many times as necessary to generate a whole row of values. for example. 'member' specifies loads and geometry (length, unbraced length, moment, etc.). 'shape' contains cross-sectional properties of the cross-section. 'matl' is material properties. Default value is set to C300W."

```
self.classNumber = self.class11p2(member=member, shape=shape)
Fy = matl.Fy(shape)
```

```
#####
```

```
# Sub Routin Start
```

```
#aShape and aMember are the shape and member objects in Sub Routin
```

```
#####
```

```
#Make mr13p6 a local variable, initialized from nr13p6 at the same 'level' as Mr
```

```
def omega2( self=self, aMember=member, aShape=shape ):
```

```
    """Returns 'omega2' coefficient that is taken to account
    for increased moment resistance of a laterally unsupported
    beam segment when subjected to a moment gradient.(part 2.2
    of the Handbook "Symbols"). Moment sign convention is
    considered as it described in MemberSegment module."""
```

```
    if max(aMember.Mend1, aMember.Mend2) != 0:
```

```
        sign = aMember.Mend1*aMember.Mend2/ abs(aMember.Mend1)/ abs(\
        aMember.Mend2)
```

```
        kappa = sign*min(abs(aMember.Mend1), abs(aMember.Mend2))/max(\
        abs(aMember.Mend1), abs(aMember.Mend2))
```

```
    else:
```

```
        kappa = 0.
```

```
    if abs(aMember.Mmax) > max(abs(aMember.Mend1), abs(aMember.Mend2)):
```

```
        omega2 = 1.0
```

```
    elif (aMember.SpEnd1 == 'false' or aMember.SpEnd2 == 'false'):
```

```
        omega2 = 1.0
```

```
    else :
```

```
        omega2 = 0.6+0.4*kappa
```

```
    return omega2
```

```
def mr13p6( self, Phi=0.9, aMember=member, aShape=shape ):
```

```
    """Returns Mr the factored moment resistance developed by a member
    subjected to uniaxial bending moments about a principal axis and
    where continuous lateral support is "not" provided to the
    compression flange. for class 1, 2 and 3 sections. 'Phi' is the
    resistance factor. Default is set to .9"""
```

```
    pi = 3.1415926
```

```
    omegaValue = omega2(aMember=aMember, aShape=aShape)
```

```
    Mu = (pi/omegaValue/aMember.UnbracedLength)*sqrt( aShape.E*\
        aShape.Iy*aShape.G*aShape.J + aShape.Iy*aShape.Cw*(pi*aShape.E\
        /aMember.UnbracedLength)**2.)
```

```
    if self.classNumber in [1,2]:
```

```
        Mp = Fy*aShape.Zx
```

```
        if Mu > 0.67*Mp:
```

```
            Mr = min( 1.15*Phi*Mp*(1.-0.28*Mp/Mu), Phi*Mp )
```

```
        else:
```

```
            Mr = Phi*Mu
```

```
    elif self.classNumber == 3:
```

```
        My = Fy*aShape.Sx
```

```
        if Mu > 0.67*My:
```

```
            Mr = min( 1.15*Phi*My*(1.-0.28*My/Mu), Phi*My )
```

```

        else:
            Mr = Phi*Mu
    else :
        MyError ="Class 4 (Slender section)\n\
        Mr should be calculated\n\
        by applying other codes"
        raise MyError
    return Mr;

def Mp( self):
    """Returns 'Mp' max moment capacity of section called plastic moment
    """
    if self.classNumber in [1,2]:
        Mp = Fy*aShape.Zx
    elif self.classNumber == 3:
        Mp = Fy*aShape.Sx
    else :
        MyError ="Class 4 (Slender section)\n\
        Mr should be calculated\n\
        by applying other codes"
        raise MyError
    return Mp

#Make mr13p6 a local variable, initialize mr13p6 at the same 'level' as Mr
def Mr( self=self, mr13p6=mr13p6, aMember=member, aShape=shape ):
    """ Returns Mr (Maximum bending resistance)"""
    if self.classNumber > 3:
        return 0
    if aMember.L < aMember.UnbracedLength:
        LengthError ="Member Length Error\n\
        Length should be less than UnbracedLength."
        raise LengthError
    if aMember.L > aMember.UnbracedLength:
        self.Mr = mr13p6(self)
    else :
        self.Mr = self.class13p5(aMember, aShape)
    return (self.Mr)/ 1E06
#####
# Sub Routin End
#####
dic = {}
dic["Mr"] = Mr(self)
dic["w2"] = omega2(self)
return dic
#####

def clause11p2( self, member, shape, matl=C300W ):
    """Returns the class number of the section.
    'member' specifies loads and geometry (length, unbraced length,
    moment, etc.). 'shape' contains cross-sectional properties
    of the cross-section, 'matl' is material properties.
    Default value is set to C300W."""
    Fy = matl.Fy(shape)
    Cy = Fy*(shape.A)

```

```

if shape.BT <= (145./(Fy**0.5)) and \
    shape.HW < ((1100.*(1.-1.40*member.Cf/Cy))/(Fy**0.5)):
    return 1
elif shape.BT <= (170./(Fy**0.5)):
    if member.Cf/Cy <= 0.15:
        shape.HW < ((1370.*(1.-1.28*member.Cf/Cy))/(Fy**0.5))
    else :
        shape.HW < ((1180.*(1.-0.43*member.Cf/Cy))/(Fy**0.5))
    return 2
elif shape.BT <= (260./(Fy**0.5)):
    if member.Cf/Cy <= 0.15:
        shape.HW < ((1810.*(1.-1.69*member.Cf/Cy))/(Fy**0.5))
    else :
        shape.HW < ((1470.*(1.-0.54*member.Cf/Cy))/(Fy**0.5))
    return 3
return 4

```

```

def clause13p5( self, member, shape, matl=C300W, Phi=0.9 ):
    """Returns Mr the factored moment resistance developed
    by a member subjected to uniaxial bending moments about
    a principal axis and where continuous lateral support is
    provided to the compressive flange for class number 1,
    2 and 3 sections.
    'member' specifies loads and geometry (length, unbraced length,
    moment, etc.). 'shape' contains cross-sectional properties
    of the cross-section. 'matl' is material properties.
    Default value is set to C300W. 'Phi' is the resistance factor.
    Default is set to .9"""
    Fy = matl.Fy(shape)
    self.classNumber = self.class11p2(member, shape)
    if self.classNumber in [1,2]:
        Mr = Phi*Fy*shape.Zx
    elif self.classNumber == 3:
        Mr = Phi*Fy*shape.Sx
    else :
        MyError = "Class 4 (Slender sections)\n\
        Mr should be calculated\n\
        by applying other codes"
        raise MyError
    return Mr

```

```

def cr13p3( self, member, shape, matl=C300W, Phi=0.9, nn=1.34 ):
    """Returns Cr -the factored compressive resistance of a
    member or component; acting at the centroid of that part
    of the steel area in compression for class number 1,2 and 3 sections.
    'member' specifies loads and geometry (length, unbraced length,
    moment, etc.). 'shape' contains cross-sectional properties
    of the cross-section. 'matl' is material properties.
    Default value is set to C300W. 'Phi' is the resistance factor.
    Default is set to .9, 'nn' is a parameter for compressive resistance.
    Default is set to 1.34."""
    Fy = matl.Fy(shape)
    pi = 3.1415926
    K1 = member.L/shape.Rx

```

```

K2 = member.L/shape.Ry
Kmax = max( K1, K2)
lanbda = (K*Kmax)*sqrt(Fy/shape.E/(pi**2.))
if self.classNumber in [1,3]:
    if lanbda <= 0.15:
        Cr = Phi * shape.A * Fy
    elif lanbda <= 1.0:
        Cr = Phi * shape.A * Fy* (1.035-0.202*lanbda-0.222*lanbda**2.)
    elif lanbda <= 2.0:
        Cr = Phi * shape.A * Fy* (-0.111+0.636*lanbda**(-1.)+0.087*lanbda**(-2.))
    elif lanbda <= 3.6:
        Cr = Phi * shape.A * Fy* (0.009+0.877*lanbda**(-2.))
    else:
        Cr = Phi * shape.A * Fy* lanbda**(-2.)
else :
    MyError = "Class 4 (Slender section)\n\
    Mr should be calculated\n\
    by applying other codes"
    raise MyError
return Cr;

```

```

def vr13p4p1p1( self, member, shape, matl=C300W, Phi=0.9, a=0.0 ):

```

```

    """Returns 'Vr' the factored shear resistance developed by
    the web of flexural member., for elastic analysis - Sec. 13.4.1.1.
    'member' specifies loads and geometry (length, unbraced length,
    moment, etc.). 'shape' contains cross-sectional properties
    of the cross-section. 'matl' is material properties.
    Default value is set to C300W. 'Phi' is the resistance factor.
    Default is set to .9, 'a' is the distance between stiffeners.
    Default is set to 0 meaning there is no stiffeners."""

```

```

    Fy = matl.Fy(shape)
    h = shape.D - 2.*shape.T
    if a == 0 or member.L == 0 :                #Unstiffened web
        kv = 5.34
    elif (a/h) < 1.0:
        kv = 4.0 + 5.34/(a/h)**2.
    else:
        kv = 5.34 + 4.0/(a/h)**2.

```

```

def Fcri( self=self, kv=kv, aShape=shape ):
    return 290.*sqrt(kv*Fy)/aShape.HW

```

```

def Fcre( self=self, kv=kv, aShape=shape ):
    return 180000.0*kv/aShape.HW**2.

```

```

def Tau( self=self, a=a, h=h):
    if a == 0 or member.L == 0:                #Unstiffened web
        return 1.0
    return (1. - 0.866/sqrt(1.0 + (a/h)**2.))

```

```

def Nu( self=self, a=a, h=h):
    if a == 0 or member.L == 0:                #Unstiffened web
        return 0.0
    return (0.5/sqrt(1.0 + (a/h)**2.))

```

```

if shape.HW <= 439.*sqrt(kv/Fy):
    Fs = 0.66*Fy
elif shape.HW <= 502.*sqrt(kv/Fy):
    Fs = Fcri()
elif shape.HW <= 621.*sqrt(kv/Fy):
    Fs = (Fcri()*Tau()) + (Nu()*Fy)
else :
    Fs = (Fcre()*Tau()) + (Nu()*Fy)
Aw = shape.D*shape.W
Vr = Phi*Aw*Fs/1000.0
return Vr

```

#This functions shows the idea of query. It demonstrates of
 #how a function could be called automatically by a method
 #for doing the calculation.

```

def calculatorFor(self, query):
    """Returns the function name which is called to generate
    the value of 'query'. 'query' could be either 'Vr' or 'Mr'."""
    if query == 'Vr':
        standard = S16p1M78()
        function = standard.vr13p4p1p1
    if query == 'Mr':
        standard = S16p1M78()
        function = standard.clause13p6
    return function

```

```

if __name__ == '__main__':
    """This is a test program that uses S16p1M87 to compute Mr, w2 for a W360x57"""
    # Assigns all attributes of tables with the name of tables header
    from SST import SST, Shp_W
    from HHBlib.IShape import IShape
    from HHBlib.MemberSegment import MemberSegment
    anInstance_SST = SST()
    anInstance_SST.registerShapeClass(IShape,['W'])
    aShape = anInstance_SST.selectByName('W360x57')
    aMember = MemberSegment(Length=19000.0, UnbracedLength= 5000., \
        Mend1=0., Mend2=0., Mmax=0., SpEnd1='false', SpEnd2='false', \
        Cf=0.0)
    standardInstance = S16p1M78()
    aDictionary = standardInstance.clause13p6(aMember, aShape)
    print "clause13p6 dictionary", aDictionary
    print "Mr", aDictionary.get("Mr", None)
    print "w2", aDictionary.get("w2", None)
    sh = anInstance_SST.selectByName(aShape)
    print "Ix", sh.Ix
    print "Vr", standardInstance.vr13p4p1p1(aMember, aShape)

```

Beam Selection Tables

Source code for Beam Selection Tables is provided in following pages.

```
#!/usr/bin/env python
```

```
"""
```

```
This module includes BST class that generates Beam Selection Tables. Beam
Selection Tables is a design aid in the Handbook of Steel Construction."""
```

```
import sys
sys.path.insert(0, "..")
from S16p1M78 import S16p1M78
from DesignParameters import DesignParameters
from SteelProperties import C300W
```

```
class BST :
```

```
    """Generates values for Beam Selection Tables cell exactly as they appear
    in the third edition of the Handbook of Steel Construction. Input
    Parameters: None. Output Parameters: None"""
```

```
    def __init__( self, member, shape_directory, UnbracedLengths, matl=C300W):
```

```
        """Initializes row and column titles of the beam selection table.
        'member' specifies loads and geometry (length, unbraced length,
        moment, etc.). 'shape_directory' is a list of cross-sections.
        UnbracedLengths is list of unbraced lengths that BST would generate
        values for. 'matl' is material properties. Default value is set to
        C300W."""
```

```
        self.aMember = member
        self.shape_directory = shape_directory
        self.UnbracedLengths = UnbracedLengths
        self.matl = matl
```

```
    def header( self , titles=["Desig. "]):
```

```
        """Returns column titles of the beam selection table.
        titles is a list of titles. Default is set to a list that has
        Desig. only"""
        titles = titles + [ "J ", "Cw ", " b ", "Lu ", "Vr ", "Ix ", "Mr " ]
        for anUnbraced_length in self.UnbracedLengths:
            #Assign anUnbraced_length to member attributes
            self.aMember.UnbracedLength = anUnbraced_length
            titles.append("%.3f" % anUnbraced_length)
        return titles
```

```
    def content( self , anInstance_standard = S16p1M78() ):
```

```
        """Returns column contents of the beam selection table.
        'anInstance_standard' is an instance of standard with S16p1M78() as
        default."""
        contents = []
        self.anInstance_standard = anInstance_standard
        #Create queries from instance of standard
        queryVr = anInstance_standard.calculatorFor('Vr')
        queryMr = anInstance_standard.calculatorFor('Mr')
        #This loop creates and stores values of shape attributes for each shape
```



```

for aShape in self.shape_directory:
    try :
        aShape.set_matl(self.matl)
    except:
        type, value, traceback = sys.exc_info()
        print aShape.Dsg , "Error in setting material:", value
        continue
    designParametersInstance = DesignParameters(self.aMember, aShape)
    contents = contents + [aShape.Dsg, \
        "%0.1f" %(aShape.J/1E03), \
        "%0.2f" %(aShape.Cw/1E09), \
        "%0.1" %(aShape.B), \
        "%0.5" %(designParametersInstance.Lu(self.aMember, aShape)), \
        "%0.1f" %queryVr(self.aMember, aShape), \
        "%0.2f" %(aShape.Ix/1E06), \
        "%0.1f" %queryMr(self.aMember, aShape)['Mr']]
    #This loop creates and stores Mr values for each unbraced lengths
    for anUnbraced_length in self.UnbracedLengths:
        #Assign anUnbraced_length to member attributes
        self.aMember.UnbracedLength = anUnbraced_length
        contents.append("%0.1f" % queryMr(self.aMember, aShape)['Mr'])
    return contents

if __name__ == '__main__':
    from SST import SST, Shp_W, Shp_WWF
    from HHBlib.IShape import IShape
    from HHBlib.MemberSegment import MemberSegment
    anInstance_SST = SST()
    anInstance_SST.registerShapeClass(IShape,['W','WWF'])
    #Extract W and WWF shapes from SST database
    #Lambda filters out non-beam sections
    shape_directory = filter(lambda s: s.Uses!=1, anInstance_SST.select(n=384,\
        Shps=[Shp_W,Shp_WWF]))
    UnbracedLengths = [2000., 2500., 3000., 3500., 4000., 4500., 5000., 5500., 6000.,\
        7000., 8000., 9000., 10000., 12000., 14000.]
    #Assign a value greater than any unbraced length that table is generated for
    #to make length greater than unbraced length
    UpperLimitUnbracedLength = 25000.
    aMember = MemberSegment(Length=UpperLimitUnbracedLength, UnbracedLength=\
        5000., Mend1=0., Mend2=0., Mmax=0., SpEnd1='false', SpEnd2='false', \
        Cf=0.0)
    aBST = BST(aMember, shape_directory, UnbracedLengths, matl=C300W)
    print aBST.header()
    print aBST.content()

```

Beam Load Tables

Source code for Beam Load Tables is provided in following pages.

```
#!/usr/bin/env python

"""
This module includes BLT class that generates Beam Load Tables. Beam
Selection Tables is a design aid in the Handbook of Steel Construction.
"""

import sys
sys.path.insert(0, "..")
from S16p1M78 import S16p1M78
from SteelProperties import C300W

class BLT :
    """Generates values for Beam Load Tables cell exactly as they appear
    in the third edition of the Handbook of Steel Construction. Input
    Parameters: None. Output Parameters: None"""

    def __init__( self, member, shape_directory, UnbracedLengths, matl=C300W):
        """Initializes row and column titles of the Beam Load Table.
        'member' specifies loads and geometry (length, unbraced length,
        moment, etc.). 'shape_directory' is a list of cross-sections.
        UnbracedLengths is list of unbraced lengths that BLT would generate
        values for. 'matl' is material properties. Default value is set to
        C300W."""
        self.aMember = member
        self.shape_directory = shape_directory
        self.UnbracedLengths = UnbracedLengths
        self.matl = matl

    def header( self, titles=["Desig. "]):
        """Returns column titles of the beam selection table.
        titles is a list of titles. Default is set to a list that has
        Desig. only"""
        titles = titles + ["Mass "]
        for anUnbraced_length in self.UnbracedLengths:
            #Assign anUnbraced_length to member attributes
            self.aMember.UnbracedLength = anUnbraced_length
            titles.append("%.3f" % anUnbraced_length)
        return titles

    def content( self, anInstance_standard = S16p1M78() ):
        """Returns column contents of the beam selection table.
        'anInstance_standard' is an instance of standard with S16p1M78() as
        default."""
        contents = []
        self.anInstance_standard = anInstance_standard
        #Create queries from instance of standard
        queryVr = anInstance_standard.calculatorFor('Vr')
        queryMr = anInstance_standard.calculatorFor('Mr')
        #This loop creates and stores values of shape attributes for each shape
        for aShape in self.shape_directory:
            try :
```

```

        aShape.set_matl(self.matl)
    except:
        type, value, traceback = sys.exc_info()
        print aShape.Dsg, "Error in setting material:", value
        continue
    contents = contents + [aShape.Dsg, "%.1f" % (aShape.Mass)]
    #This loop creates and stores Mr values for each unbraced lengths
    for anUnbraced_length in self.UnbracedLengths:
        #Assign anUnbraced_length to member attributes
        self.aMember.UnbracedLength = anUnbraced_length
        Mr = queryMr(self.aMember, aShape)['Mr']
        Vr = queryVr(self.aMember, aShape)
        UDFL1 = round (((8000.*Mr)/(anUnbraced_length)), 0)
        #UDFL stands for Uniformly Distributed Factored Loads
        UDFL2 = round ((2.*Vr), 0)
        UDFL = min(UDFL1, UDFL2)
        contents.append("%.1f" % UDFL)
    return contents

if __name__ == '__main__':
    from SST import SST, Shp_W, Shp_WWF
    from HHBlib.IShape import IShape
    from HHBlib.MemberSegment import MemberSegment
    anInstance_SST = SST()
    anInstance_SST.registerShapeClass(IShape,['W','WWF'])
    #Extract W and WWF shapes from SST database
    #Lambda filters out non-beam sections
    shape_directory = filter(lambda s: s.Uses!=1, anInstance_SST.select(n=384,\
        Shps=[Shp_W,Shp_WWF]))
    UnbracedLengths = [2000., 2500., 3000., 3500., 4000., 4500., 5000., 5500., 6000.,\
        7000., 8000., 9000., 10000., 12000., 14000.]
    #Assign a value greater than any unbraced length that table is generated for
    #to make length greater than unbraced length
    UpperLimitUnbracedLength = 25000.
    aMember1 = MemberSegment(Length=UpperLimitUnbracedLength, UnbracedLength=\
        5000., Mend1=0., Mend2=0., Mmax=0., SpEnd1='false', SpEnd2='false', \
        Cf=0.0)
    aBLT = BLT(aMember1, shape_directory, UnbracedLengths, matl=C300W)
    print aBLT.header()
    print aBLT.content()

```

Class of Sections in bending

Source code for Class of Sections in bending is provided in following pages.

```
#!/usr/bin/env python
```

```
"""
```

```
This module includes CST class that generates Class of Section in Bending
Tables. Class of Section in Bending is a design aid in the Handbook of Steel
Construction."""
```

```
import sys
sys.path.insert(0, ".")
from S16p1M78 import S16p1M78
from SteelProperties import A36, C300W, C350W
```

```
class CST :
```

```
    """Generates values for Class of Section in Bending Tables coll exactly as
    they appear in the third edition of the Handbook of Steel Construction.
    Input Parameters: None. Output Parameters: None"""
```

```
    def __init__( self, member, shape_directory, matl=C300W):
        """Initializes row and column titles of the beam selection table.
        'member' specifies loads and geometry (length, unbraced length,
        moment, etc.). 'shape_directory' is a list of cross-sections.
        'matl' is material properties. Default value is set to C300W."""
        self.aMember = member
        self.shape_directory = shape_directory
        self.matl = matl
```

```
    def header( self, titles=["Desig. "]):
        """Returns column titles of the beam selection table.
        titles is a list of titles. Default is set to a list that has
        Desig. only"""
        titles = titles + ["A36 ", " C300W", "C350W "]
        return titles
```

```
    def content( self, anInstance_standard = S16p1M78() ):
        """Returns column contents of the beam selection table.
        'anInstance_standard' is an instance of standard with S16p1M78() as
        default."""
        contents = []
        self.anInstance_standard = anInstance_standard
        #This loop creates and stores values of shape attributes for each shape
        for aShape in self.shape_directory:
            contents.append(aShape.Dsg)
            #This loop calculates class number
            #for different assigned steel properties of aShape
            for matl in [A36, C300W, C350W]:
                try :
                    aShape.set_matl(self.matl)
                except:
                    type, value, traceback = sys.exc_info()
                    print aShape.Dsg , "Error in setting material:", value
                    continue
```

```

        contents.append( self.anInstance_standard.clause11p2( self.aMember, aShape, matl))
    return contents

if __name__ == '__main__':
    from SST import SST, Shp_W, Shp_WWF
    from HHBlib.IShape import IShape
    from HHBlib.MemberSegment import MemberSegment
    anInstance_SST = SST()
    anInstance_SST.registerShapeClass(IShape,['W','WWF'])
    #Extract W and WWF shapes from SST database
    #Lambda filters out non-beam sections
    shape_directory = filter(lambda s: s.Uses!=1, anInstance_SST.select(n=185,\
        Shps=[Shp_W,Shp_WWF]))
    aMember = MemberSegment(Length=8000., UnbracedLength= \
        5000., Mend1=0., Mend2=0., Mmax=0., SpEnd1='false', SpEnd2='false', \
        Cf=0.0)
    aCST = CST(aMember, shape_directory, matl=C300W)
    print aCST.header()
    print aCST.content()

```

Design Parameters

Source code for Design Parameters is provided in following pages.

```
#!/usr/bin/env python

"""
This module computes design parameters that is not driven from the formulas
specified in the Standard as they appear. This module is based on S16p1M78."""

import sys
sys.path.insert(0, "..")
from S16p1M78 import S16p1M78
from HHBlib.Utils import maxX4maxY

class DesignParameters:
    """This class returns design parameters like Lu. Input Parameters: None.
    Output Parameters: None"""

    def __init__( self , member, shape):
        """Initializes a member and a shape."""
        self.member = member
        self.shape = shape

    def Lu( self, member, shape):
        """Compute max unsupported beam length for which Mr is applicable (mm)
        . Mr is factored moment resistance for laterally supported member
        (kN.m)."""
        standardInstance = S16p1M78()
        queryMr = standardInstance.calculatorFor('Mr')

    def getMr( length, queryMr=queryMr, member=member, shape=shape):
        """Returns Mr that is factored moment resistance for laterally
        supported member (kN.m)"""
        member.UnbracedLength = length
        return queryMr(member, shape)['Mr']

    #Assign a minimum UnbracedLength (0.0 causes floating point error)
    member.UnbracedLength = 0.001
    #Range of function is UnbracedLength to L
    LuValue = maxX4maxY( getMr, member.UnbracedLength, member.L, 0.000001,\
        0.000001);
    return "%i" % LuValue

if __name__ == '__main__':
    """This is a test program that uses S16p1M87 to compute Lu for a
    W360x57"""
    # Assigns all attributes of tables with the name of tables header
    from SST import *
    from HHBlib.MemberSegment import *
    from HHBlib.IShape import *
    anInstance_SST = SST()
    anInstance_SST.registerShapeClass(IShape,['W'])
    aShape = anInstance_SST.selectByName('W360x57')
```

```
aMember= MemberSegment(Length=15000.0, UnbracedLength= 3500.0, \
Mend1=0., Mend2=0., Mmax=0., SpEnd1='false', SpEnd2='false', Cf=0.0)
anInstance = DesignParameters(aMember, aShape)
print "Shape:", aShape, "UnbracedLength:", aMember.UnbracedLength
print "Lu:", anInstance.Lu(aMember, aShape)
```

Member Segment

Source code for Member Segment is provided in following pages.

```
#!/usr/bin/env python

"""This module specifies loads and geometry (length, unbraced length, moment,
etc) for a steel member segment.

***** Sign Convention *****
The sign convention used in the beam section is defined in the following
illustrations. (Please refer to Figure )
Notice that unlike some publications on this subject, the loading term P(x)
points to the same direction as y-axis.
Please refer to the Euler-Bernoulli beam theory for more details."""

class MemberSegment:
    """Represents a segment of a member. Each segment of a member is identified
    by a certain geometry condition or load condition. Input Parameters: None.
    Output Parameters: None"""

    def __init__( self , Length=0.0, UnbracedLength=6000., Mend1=0, Mend2=0.,
                  Mmax=0., SpEnd1='false', SpEnd2='false', Cf=0.0):
        """Initializes segment specifications (loads and geometry). Length is
        member's length. Default is set to 0.0. UnbracedLength is the length of
        segment which defined by it's braced barriers. Default is set to 6000.,
        Mend1 is the moment at end 1 of segment. Default is set to 0.0,
        Mend2 is the moment at end 2 of segment. Default is set to 0.0,
        Mmax is the max moment between end 1 to 2, but only if greater than end
        moments. Default is set to 0.0, SpEnd1 and SpEnd2 are support booleans
        at end 1 and 2 respectively , which are true if comp flange are
        supported at that end. Default is set to 'false'. Cf is the axial force
        that acts on the member. Default is set to 0.0, meaning that there is no
        axial force."""
        self.L = Length
        self.UnbracedLength = UnbracedLength
        self.Mend1 = float( Mend1 )
        self.Mend2 = float( Mend2 )
        self.Mmax = float( Mmax )
        self.SpEnd1 = SpEnd1
        self.SpEnd2 = SpEnd2
        self.Cf = Cf

if __name__ == '__main__':

    p=MemberSegment( 4000., 3000., 250., 500.)

    print "Length", p.L
    print "Unbraced Length", p.UnbracedLength
    print "Mend1", p.Mend1
    print "Mend2", p.Mend2
```

Steel Properties

Source code for Steel Properties is provided in following pages.

STEEL

SteelProperties module set default materials for all section types, and will allow an application program to set different default. All strength calculations use this module to get Fy, Fu, E and G unless these have been specified explicitly when the method is called.

SteelProperties module is a computerized version of steel properties table in the Handbook that assigns all steel properties. These properties are exactly same as what is in Table 6-1 to 6-3 in 3rd Edition of the Handbook. ASTM properties is also added due to the fact that other design aids such Class of Section in bending is using it.

The SteelProperties module defines the following classes:

SteelBase

Class SteelBase assigns nominal Fy, Fu, E and G. Methods exist to retrieve nominal Fy, Fu, E and G one by one or whole as a dictionary for a shape by its designation.

G4021

Class G4021 inherits SteelBase and it is a super Class too.

G4021_300W

Class G4021_300W instances contain the steel properties for steel of grade 300W. It is an inheritant of G4021.

G4021_350W

Class G4021_300W instances contain the steel properties for steel of grade 350W. It is an inheritant of G4021.

ASTM

Class ASTM inherits SteelBase and it is a super Class too.

A36

Class A36 instances contain the steel properties for steel of grade A36. It is an inheritant of ASTM.

GradeTable

Class GradeTable provides get and set methods. It can be called by its abbreviation name "Grade" too.

GroupExceedsMaximum

This exception is raised when a shape specified group is not found in the mechanical steel properties table.

ThicknessExceedsMaximum

This exception is raised when a shape specified thickness is greater Nominal Max thickness in the mechanical steel properties table.

The module has a test program at the end of it that test the module by assigning different grades to different shapes. """

```
from SST import Shp_WWF, Shp_WWT, Shp_WRF, Shp_HSS_SQ, Shp_HSS_RCT, Shp_HSS_RND
```

```
__ID = """Std: SteelProperties.py,v 1.2 2003/04/10 00:02:05 bansari Exp S"""
```

```
class GroupExceedsMaximum(Exception): pass
```

```
class ThicknessExceedsMaximum(Exception): pass
```

```
class SteelBase:
```

```
    """Assigns nominal Fy, Fu, E and G. Methods exist to retrieve nominal Fy,
    Fu, E and G one by one or whole as a dictionary for a shape by its
    designation. Input Parameters: None. Output Parameters: None"""
```

```
    def __init__( self, name, nominalFy, nominalFu, nominalE=200000.0,
                  nominalG=77E3 ):
```

```
        """Initializes nominal value of basic steel properties in the order of
        'table name'. Fy, Fu, E and G. Default nominal values are:
        nominalE=200000.0 and nominalG=77E3."""
```

```
        self.name = name
        self.nominalFy = nominalFy
        self.nominalFu = nominalFu
        self.nominalE = nominalE
        self.nominalG = nominalG
```

```
    def Fy( self, shape=None ):
```

```
        """Returns actual Fy for a specific shape. 'shape' is the steel section
        designation. Default is set to None which means nominal values would be
        assigned, Fy=350.0"""
```

```
        return self.nominalFy
```

```
    def Fu( self, shape=None ):
```

```
        """Returns the Fu for a specific shape. 'shape' is the steel section
        designation. Default is set to None which means nominal values would be
        assigned, Fu=450.0"""
```

```
        return self.nominalFu
```

```
    def E( self, shape=None ):
```

```
        """Returns the E for a specific shape. 'shape' is the steel section
        designation. Default is set to None which means nominal values would be
        assigned, nominalE=200000.0"""
```

```
        return self.nominalE
```

```
    def G( self, shape=None ):
```

```
        """Returns the G for a specific shape. 'shape' is the steel section
        designation. Default is set to None which means nominal values would be
        assigned, nominalG=77E3"""
```

```
        return self.nominalG
```

```
    def properties( self, shape=None, n=4 ):
```

```
        """Returns a tuple of up to 4 properties in the order Fy, Fu, E and G.
        'shape' is the steel section designation. Default is set to None which
        means nominal values would be assigned, Fy=350.0, Fu=450.0,
        nominalE=200000.0, nominalG=77E3. 'n' is number of sections that should
        be selected. Default is set to 4"""
```

```
        p=(self.Fy(shape),self.Fu(shape),self.E(shape),self.G(shape))
        return p[0:n]
```

```

def __str__( self):
    """Return a string suitable for printing."""
    return "[Steel %s]"%( self.name )

def __repr__( self):
    """Produce the string representation of the matl object."""
    return "<%s: %s>"%( self.__class__.__name__, self.__str__())

```

```

class G4021( SteelBase ):

```

```

    """This class implements the material property calculations for CSA
    G40.21. Input Parameters: None. Output Parameters: None"""

```

```

def __init__( self, name, maxt, Fyt1, Fyt2, Fyt3, maxGrp, Fyg3and4, Fyg5,\
    maxwt, Fyhss, Fy=350.0, Fu=450.0):
    """Initializes column titles of the Steel Properties table. Default
    values are: Fy=350.0, Fu=450.0"""
    self.name=name
    self.maxt=maxt
    self.Fyt1=Fyt1
    self.Fyt2=Fyt2
    self.Fyt3=Fyt3
    self.maxGrp=maxGrp
    self.Fyg3and4=Fyg3and4
    self.Fyg5=Fyg5
    self.maxwt=maxwt
    self.Fyhss=Fyhss
    SteelBase.__init__( self, name, Fy, Fu)

def Fy( self, shape=None ):
    """Returns actual Fy for a specific shape. 'shape' is the steel
    section designation. Default is set to None which means nominal
    values will be assigned, Fy=350.0"""
    if shape is None:
        return SteelBase.Fy(self)
    #for WWF (Welded) shapes
    elif shape.Shp in [Shp_WWF, Shp_WWT, Shp_WRF]:
        if max(shape.W, shape.T ) > self.maxt:
            raise ThicknessExceedsMaximum("Thickness exceeds maximum " + \
            shape.Dsg )
        if max(shape.W, shape.T ) <= 40.0:
            return self.Fyt1
        elif max(shape.W, shape.T ) <= 65.0:
            return self.Fyt2
        else:
            return self.Fyt3
    # Hollow Structural Sections
    elif shape.Shp in [Shp_HSS_SQ, Shp_HSS_RCT, Shp_HSS_RND]:
        return self.Fyhss
    elif shape.Grp > self.maxGrp:
        raise GroupExceedsMaximum("Group exceeds maximum for shape " + \
        shape.Dsg )
    elif shape.Grp <= 2:
        return self.nominalFy
    elif (shape.Grp == 3 or shape.Grp == 4):

```

```

        return self.Fyg3and4
    elif shape.Grp == 5:
        return self.Fyg5
    else:
        raise ("Cannot assign Fy to " + shape.Dsg)

class G4021_300W( G4021 ):
    """Returns a row of table with 300W grade character.
    Default calculations just return nominal values of G4021 by inheritance.
    Input Parameters: None. Output Parameters: None"""

    def __init__( self ):
        """Initializes row values for each column title of the Steel Properties
        table."""
        G4021.__init__(self, name="CSA G40.21 300W", maxt=100.0, Fyt1=300.0,\
            Fyt2=290.0, Fyt3=280.0,maxGrp=3, Fyg3and4=290.,\
            Fyg5=None, maxwt=16.0, Fyhss=300., Fy=300.0, Fu=450.0)

class G4021_350W( G4021 ):
    """Returns a row of table with 350W grade character. Default calculations
    just return nominal values of G4021 by inheritance. Input Parameters: None.
    Output Parameters: None"""

    def __init__( self ):
        """Initializes row values for each column title of the Steel Properties
        table."""
        G4021.__init__(self, name="CSA G40.21 350W", maxt=100.0, Fyt1=350.0,\
            Fyt2=330.0, Fyt3=320.0, maxGrp=2, Fyg3and4=None,\
            Fyg5=None, maxwt=16.0, Fyhss=350., Fy=350.0, Fu=450.0)

class G4021_300WT( G4021 ):
    """Returns a row of table with 300WT grade character. Default calculations
    just return nominal values of G4021 by inheritance. Input Parameters: None.
    Output Parameters: None"""

    def __init__( self ):
        """Initializes row values for each column title of the Steel Properties
        table."""
        G4021.__init__(self, name="CSA G40.21 300WT", maxt=100.0, Fyt1=300.0,\
            Fyt2=290.0, Fyt3=280.0, maxGrp=5, Fyg3and4=290.0,\
            Fyg5=280.0, maxwt=None, Fyhss=None, Fy=300.0, Fu=450.0)

class ASTM( SteelBase ):
    """This class implements the material property calculations for ASTM"""

    def __init__(self, name, Fy1, Fy=350.0, Fu=450.0):
        """Initializes nominal value of basic steel properties in the order
        of "table name", Fy1, Fy and Fu. Fy1 is another nominal value
        assigned for Fy for ASTM steel. Default values are: Fy = 350.0 and
        Fu = 450.0. "table name", Fy and Fu initialized by inheritance from
        SteelBase class."""
        self.name = name
        self.Fy1 = Fy1

```

```

SteelBase.__init__(self, name, Fy, Fu)

def Fy( self, shape=None ):
    """Returns actual Fy for a specific shape. 'shape' is the steel
    section designation. Default is set to None which means assigned
    value will be nominal value defined in SteelBase class"""
    if shape is None:
        return SteelBase.Fy( self )
    else:
        return self.Fy1

class A36(ASTM):
    """This class implements the material property calculations for ASTM A36.
    Default calculations just return nominal values of name and Fy1 from ASTM
    by inheritance."""

    def __init__( self ):
        """Initializes "table name" to "ASTM A36" and Fy1 to 250 in ASTM class
        """
        ASTM.__init__(self, "ASTM A36", 250.)

class GradeTable:
    """This class maintains default steel grades for each type of section.
    Input Parameters: None. Output Parameters: None"""

    def __init__( self ):
        """Initializes an empty dictionary."""
        self.table = {}

    def setSteel( self, shpCode, steelGrade ):
        """Sets the default grade of steel to use for shapes with given shape
        designation code. 'shpCode' is assigned with each steel section
        designation. 'steelGrade' is the grade of mechanical propertie
        of the steel."""
        self.table[shpCode] = steelGrade

    def getSteel( self, shpCode ):
        """Return the default grade of steel for this shape. 'shpCode' is the
        code that is assigned with each steel section designation."""
        return self.table[shpCode]

Grades = GradeTable()

# create a couple of common grades of steel
C350W = G4021_350W()
C300W = G4021_300W()
C300WT = G4021_300WT()
A36 = A36()

# assign these as default grades for the various kinds of shape.
Grades.setSteel( 1, C300W )      # W shapes
Grades.setSteel( 2, C300W )      #
Grades.setSteel( 3, C300W )      #

```

```

Grades.setSteel( 4, C300W )      #

Grades.setSteel( 6, C300W )      #
Grades.setSteel( 7, C300W )      #
Grades.setSteel( 8, C300W )      #
Grades.setSteel( 9, C300W )      #

Grades.setSteel( 11, C300W )     #
Grades.setSteel( 12, C300W )     #
Grades.setSteel( 13, C300W )     #

Grades.setSteel( 15, C300W )     #
Grades.setSteel( 16, C300W )     #
Grades.setSteel( 17, C300W )     #
Grades.setSteel( 18, C300W )     #

Grades.setSteel( 5, C350W )      # WWF shapes
Grades.setSteel( 10, C350W )
Grades.setSteel( 14, C350W )

if __name__ == '__main__':
    print __ID__, "\n"
    import sys
    sys.path.insert(0, "..")
    from HHBlib.IShape import *
    from SST import SST
    sst = SST()
    sst.registerShapeClass(IShape,['W','WWF', 15, 16, 17])

    def test (dsg, matl=None, sst=sst):
        """ A test program that prints material properties for shapes if there
            is any and prints exception if there is not"""
        try:
            shape = sst.selectByName( dsg )
            shape.set_matl( matl )
        except Exception, e:
            print "Shape:", dsg, " Exception:", e
        else:
            if shape.Fy == None:
                print "There is no " "Shape:", dsg, "with Matl:", matl
            else:
                print "Shape:", dsg, " Matl:", matl, " Fy=%s Fu=%s" %\
                    (shape.Fy, shape.Fu)

    #Main part of test. Test for various sections
    for dsg in ['W610x82', 'W200x86', 'W490x289', 'W690x802', 'W920x1262', \
        'WWF900x169', 'WWF650x739', 'HS305x203x13']:
        test(dsg, C350W )
        test(dsg, C300W)
        test(dsg, C300WT)
    print

```

XML Table

Source code for XML Table is provided in following pages.

```
#!/usr/bin/env python

"""
This module provides tools to generate tables in XML format.
"""

from xml.dom.DOMImplementation import implementation

class Page:
    """Generates a document page (i.e. a table) in XML format.
    Input Parameters: None. Output Parameters: None"""

    def __init__( self ):
        """Initializes document and a table."""
        self.document = implementation.createDocument( None, None, None )
        self.table = self.document.createElement( 'TABLE' )
        self.document.appendChild( self.table )

    def addRow( self, values, emph=[] ):
        """Creates a row of a table. Row's CELL are elements of "values" list.
        Attributes of the elements that are in the emph list too get set to
        ('EMPH', '1'). emph default is an empty list."""
        doc = self.document
        row = doc.createElement( 'ROW' )
        self.table.appendChild( row )
        for i in range(len(values)):
            cell = doc.createElement( 'CELL' )
            if i in emph:
                cell.setAttribute( 'EMPH', '1' )
            row.appendChild( cell )
            cell.appendChild( doc.createTextNode( str(values[i]) ) )

    def pprint( self ):
        """Prints the document."""
        import xml.dom.ext
        xml.dom.ext.PrettyPrint( self.document )

if __name__ == '__main__':
    from SST import SST, Shp_W, Shp_WWF
    import sys
    sys.path.insert(0, "..")
    from HHBlib.IShape import IShape
    from HHBlib.MemberSegment import MemberSegment
    from SteelProperties import C300W
    anInstance_SST = SST()
    anInstance_SST.registerShapeClass(IShape,['W','WWF'])
    #Extract W and WWF shapes from SST database
    #Lambda filters out non-beam sections
    shape_directory = filter(lambda s: s.Uses!=1, anInstance_SST.select(n=1,\
        Shps=[Shp_W,Shp_WWF]))
    UnbracedLengths = [2000., 2500., 3000., 3500., 4000., 4500., 5000., 5500., 6000.,\

```

```

7000., 8000., 9000., 10000., 12000., 14000.]
#Assign a value greater than any unbraced length that table is generated for
#to make length greater than unbraced length
UpperLimitUnbracedLength = 25000.
aMember = MemberSegment(Length=UpperLimitUnbracedLength, UnbracedLength= \
    5000., Mend1=0., Mend2=0., Mmax=0., SpEnd1='false', SpEnd2='false', \
    Cf=0.0)
from BeamSelectionTable import BST
oneBST = BST(aMember, shape_directory, UnbracedLengths, matl='C300W')
page = Page()
page.addRow(oneBST.header())
page.addRow(oneBST.content())
page.pprint()

```

Utils

Source code for Utils is provided in following pages.

```
#!/usr/bin/env python
```

```
"""
```

```
This module provides calculation tools that is needed during the design with  
Canadian Steel Code."""
```

```
def maxX4maxY(input_function, lowerBound, upperBound,\
              inputFunctionTolerance=0.01, Xtolerance=0.01):
    """This function returns X Max corresponding to Max of input_function(x)
    value, given a function in input_function(X) format. 'input_function' is
    a function to find the Max X of Max [input_function(x)]. 'lowerBound' is
    lower bound on X. 'upperBound' is upper bound on X. 'inputFunctionTolerance'
    is the input_function's tolerance value. Default tolerance value is set to
    .01 'Xtolerance' is the input_function's X tolerance value. Default
    tolerance value is set to .01."""
    lowerBound = float(lowerBound)
    upperBound = float(upperBound)
    lowerBoundInputFunctionVal = input_function(lowerBound)
    upperBoundInputFunctionVal = input_function(upperBound)

    while abs(upperBound - lowerBound) > Xtolerance :

        #Assume Max_X is at midpoint
        Max_X = (upperBound + lowerBound)/2
        MaxXInputFunctionVal = input_function(Max_X)
        if abs(MaxXInputFunctionVal) <= inputFunctionTolerance :
            break

        #Check if reached desired precision then result achieved
        if abs(MaxXInputFunctionVal - lowerBoundInputFunctionVal) <= \
            inputFunctionTolerance :
            (lowerBound, lowerBoundInputFunctionVal) = \
            (Max_X, MaxXInputFunctionVal)
        else :
            (upperBound, upperBoundInputFunctionVal) = \
            (Max_X, MaxXInputFunctionVal)
    return Max_X
```

```
#A simple main to test the module
```

```
if __name__ == "__main__":
```

```
#Three function defined and used as sample input functions
```

```
def sample_input_function1(x):
```

```
    """Returns Min of (maxval,1/x), maxval is 0.5 if x value is negative.
    'X' is input value."""
```

```
    maxval = 0.5
```

```
    if x <= 0.0:
```

```
        return maxval
```

```
    return min(maxval,1.0/x)
```

```

def sample_input_function2(x):
    """Returns Min of (1/x,1), 'X' is input value."""
    return min((1/x), 1.0)

def sample_input_function3(x):
    """Returns Min of (1/x-1) and 1, 'X' is input value."""
    return min((1/x-1), 1.0)

#Execute maxX4maxY with sample_input_function1
Max_X_sampleInputFunction1 = maxX4maxY(sample_input_function1, 0.0, 1000.0\
, 0.000001, 0.000001);
print "Max_X_sampleInputFunction1 = %g" % Max_X_sampleInputFunction1

#Execute maxX4maxY with sample_input_function2
Max_X_sampleInputFunction2 = maxX4maxY(sample_input_function2, 0.05, 1000.0\
, 0.000001, 0.000001);
print "Max_X_sampleInputFunction2 = %g" % Max_X_sampleInputFunction2

#Execute maxX4maxY with sample_input_function3
Max_X_sampleInputFunction3 = maxX4maxY(sample_input_function3, 0.08, 1000.0\
, 0.000001, 0.000001);
print "Max_X_sampleInputFunction3 = %g" % Max_X_sampleInputFunction3

```

IShape

Source code for IShape is provided in following pages.

```
#!/usr/bin/python

"""Module IShape provides all attributes of an IShape from a Structural
Shape Table (SST) Database."""

import sys
sys.path=['/users/bansari/public_html']+sys.path
sys.errout = sys.stdout
from SST import Shape
from SteelProperties import Grades

class IShape(Shape):

    """An IShape is one structural section from the shape tables
    (i.e., one complete row from the table). The properties are
    stored in attributes of the correct name."""

    def __init__( self, row=[], columns=[], matl=None ):
        """Return an instance, with all attributes initialized from a row
        of the SST. 'row' and 'columns' are row and columns of the SST.
        Defaults are empty lists. 'matl' matl is the mechanical properties.
        Default is set to None which means nominal values would be assigned,
        Fy=350.0, Fu=450.0, nominalE=200000.0, nominalG=77"""
        for i in range(len(columns)):
            colname = columns[i][0]
            setattr( self, colname, row[i] )
            print vars(self)
        self.set_matl(matl)

    def set_matl(self, matl=None):
        """Set the material type and material properties of the shape. 'matl' is
        the mechanical properties. Default is set to None which means
        nominal values would be assigned, Fy=350.0, Fu=450.0, nominalE=200000.0,
        nominalG=77"""
        if matl is None:
            matl=Grades.getSteel(self.Shp)
        self.matl=matl
        self.Fy,self.Fu,self.E,self.G=matl.properties(shape=self,n=4)
        #Check to make sure the Fy value is properly set
        assert (self.Fy is not None and self.Fy) > 0.0

if __name__ == '__main__':
    from SST import *
    sst = SST()
    sst.registerShapeClass(IShape, [1])
    def selectByName( desig ):
        """Assigns attributes from SST to a 'desig' shape"""
        return sst.selectByName( desig )
    aShape=selectByName( 'W310x28' )
```

```
print "Shape: ", aShape.Dsg  
print "Attributes Dictionary: ", vars(aShape)
```