Final Research Report
Agreement T2695, Task 37
Earthquake Bridge Inspection

Information Tools to Improve Post-Earthquake Prioritization of WSDOT Bridge Inspections

by

Stephen Malone, Research Professor
Dept. of Earth and Space Sciences

Marc O. Eberhard, Professor
Dept. of Civil and Environmental Engineering

Jay LaBelle, Programmer
Dept. of Earth and Space Sciences

Tyler Ranf, Graduate Research Assistant
Dept. of Civil and Environmental Engineering

University of Washington
Seattle, Washington 98195

Washington State Transportation Center (TRAC)
University of Washington, Box 354802
1107 NE 45th Street, Suite 535
Seattle, Washington 98105-4631

Washington State Department of Transportation Technical Monitor
Harvey Coffman, Bridge Preservation Engineer
Bridge and Structures

Prepared for

Washington State Transportation Commission
Department of Transportation
and in cooperation with
U.S. Department of Transportation
Federal Highway Administration

June 2005

TECHNICAL REPORT STANDARD TITLE PAGE

1. REPORT NO. WA-RD 602.1	2. GOVERNMENT ACCESSION NO.	3. RECIPIENT'S CATALOG NO.
4. TITLE AND SUBTITLE INFORMATION TOOLS TO IMPROVE POST-EARTH-QUAKE PRIORITIZATION OF WSDOT BRIDGE INSPECTIONS		5. REPORT DATE June 2005 6. PERFORMING ORGANIZATION CODE
7. AUTHOR(S) Malone, Stephen; Eberhard, Marc. O.; LaBelle, Jay; Ranf, Tyler		8. PERFORMING ORGANIZATION REPORT NO.
9. PERFORMING ORGANIZATION NAME AND ADDRESS Washington State Transportation Center (TRAC) University of Washington, Box 354802 University District Building; 1107 NE 45th Street, Suite 535 Seattle, Washington 98105-4631		10. WORK UNIT NO. 11. CONTRACT OR GRANT NO. Agreement T2695, Task 37
12. SPONSORING AGENCY NAME AND ADDRESS Research Office Washington State Department of Transportation Transportation Building, MS 47372 Olympia, Washington 98504-7372 Kim Willoughby, Project Manager, 360-705-7978		13. TYPE OF REPORT AND PERIOD COVERED Final Research Report 14. SPONSORING AGENCY CODE

15. SUPPLEMENTARY NOTES

This study was conducted in cooperation with the U.S. Department of Transportation, Federal Highway Administration.

16. ABSTRACT

University of Washington researchers developed information tools to increase the speed and efficiency of Washington State Department of Transportation (WSDOT) post-earthquake response and recovery efforts. The researchers upgraded the Pacific Northwest Seismograph Network (PNSN) ground-motion processing software to rapidly generate and disseminate “ShakeMaps,” which are maps of earthquake intensity. The researchers also implemented two procedures to estimate the likelihood of slight (or greater) bridge damage; these procedures are based on the intensity of earthquake shaking (obtained from the ShakeMaps) and on each bridge’s location, year of construction, and bridge type (obtained from the Washington State Bridge Inventory). The first procedure, developed at the University of Washington, is based on observations of bridge damage from the 2001 Nisqually earthquake. The second procedure is contained in the Federal Emergency Management Agency HAZUS software for predicting the lowest level of damage.

Shortly following an earthquake, e-mail and pager alert messages will be sent to WSDOT personnel notifying them of the preliminary earthquake magnitude and epicenter. ShakeMaps and a prioritized list of bridges (ranked by likelihood of bridge damage) will be available on a Web server at the University of Washington and will be pushed to a WSDOT FTP server.

17. KEY WORDS Bridges, earthquakes, damage, inspection, fragilities, ShakeMap	18. DISTRIBUTION STATEMENT No restrictions. This document is available to the public through the National Technical Information Service, Springfield, VA 22616

19. SECURITY CLASSIF. (of this report)	20. SECURITY CLASSIF. (of this page)	21. NO. OF PAGES	22. PRICE
None	None		

Contents

Figures

Tables

Executive Summary

During the next decade, portions of Western Washington will likely be subjected again to at least a moderate earthquake and, perhaps, a severe one. After such an event, WSDOT will be expected to rapidly inspect bridge damage in numerous locations, divert traffic from damaged structures, and restore bridges to service.

In the past, WSDOT has had little access to reliable information immediately following an earthquake. Early phoned-in damage reports may not have been reliable, and they may not have included some of the hardest hit areas, in which communication may have been disrupted. Knowing only the earthquake magnitude and location, WSDOT engineers have dispatched inspectors to bridges on the basis of the distance from the earthquake epicenter to the bridge and field reports of observed damage.

The 2001 Nisqually event demonstrated that, even for a moderate earthquake, it is inefficient to dispatch inspectors only on the basis of the earthquake magnitude, its location, and damage reports. The challenge arises, in part, from the poor correlation between epicentral distance and likelihood of damage. The ability to identify damaged bridges quickly following an earthquake would increase public safety by focusing the initial bridge inspections on the bridges most likely to have been damaged.

As part of this project, University of Washington researchers developed information tools to increase the speed of Washington State Department of Transportation (WSDOT) post-earthquake inspection, response, and recovery efforts. The researchers upgraded the Pacific Northwest Seismograph Network (PNSN) ground-motion processing software to make it possible to generate and disseminate "ShakeMaps" shortly after an earthquake. ShakeMaps are maps of earthquake intensity derived from

measurements of ground shaking and maps of the local geology (Wald et al. 1999). The researchers also implemented two procedures to estimate the likelihood of slight (or greater) bridge damage; these procedures are based on the intensity of earthquake shaking (obtained from the ShakeMaps) and on each bridge's location, year of construction, and bridge type (obtained from the Washington State Bridge Inventory). The first procedure, developed at the University of Washington, is based on observations of bridge damage from the 2001 Nisqually earthquake (Ranf et al. 2001). The second procedure is contained in the Federal Emergency Management Agency HAZUS software for predicting the lowest level of damage.

Shortly following an earthquake, e-mail and pager alert messages will be sent to WSDOT personnel notifying them of the preliminary earthquake magnitude and epicenter. ShakeMaps and a prioritized list of bridges (ranked by likelihood of bridge damage) will be available on a Web server at the University of Washington and will be pushed to a WSDOT FTP server to be downloaded for post-earthquake response planning.

CHAPTER 1

INTRODUCTION

During the next decade, portions of Western Washington are likely to be subjected again to at least a moderate earthquake and, perhaps, a severe one. After such an event, WSDOT will be expected to rapidly assess bridge damage in numerous locations, divert traffic from damaged structures, and restore bridges to service. To help plan its response and recovery efforts, WSDOT needs tools to quickly prioritize post-earthquake inspections of bridges.

In the past, WSDOT has had little access to reliable information immediately following an earthquake. The University of Washington Rapid Alert for Cascadia Earthquakes (RACE) system has been the primary source of earthquake notification; this system transmits an estimate of the earthquake magnitude and location to key WSDOT personnel by pager and e-mail service. Knowing the earthquake magnitude and location, WSDOT engineers have dispatched inspectors to bridges on the basis of the distance from the earthquake epicenter to the bridge and field reports of observed damage.

The 2001 Nisqually event demonstrated that, even for a moderate earthquake, it is inefficient to dispatch inspectors only on the basis of the earthquake magnitude, its location, and damage reports. The challenge arises, in part, from the poor correlation between epicentral distance and likelihood of damage. This poor correlation stems from the effects of event depth, fault orientation, local soil conditions, and the vulnerability of each individual bridge. For example, the 2001 Nisqually earthquake damaged far fewer bridges in the Tacoma area than in Seattle, even though Seattle was much farther from the earthquake epicenter (Ranf et al. 2001, EERI 2001). Early phoned-in damage reports also can be unreliable, and they may not include some of the hardest hit areas, in which communication may have been disrupted.

This project developed information tools to provide WSDOT with better and faster estimates of the level of ground-motion intensity at each bridge site, and estimates of bridge damage throughout the affected region. The information tools will quickly and automatically analyze the Washington State Bridge Inventory data and ground-motion data, and they will transmit maps and prioritized bridge damage estimation lists to WSDOT following a large earthquake.

These tools were developed to increase the speed and efficiency of the WSDOT emergency response and recovery efforts.

- WSDOT will be able to assign the highest priorities for inspection to clusters of bridges with high likelihoods of damage.
- For some small earthquakes, the prioritization tools may make it possible to save resources by reducing the number of bridges that need to be inspected.
- The availability of ground-motion information will assist with the analysis of bridge damage when it is found.

CHAPTER 2

IMPROVEMENTS IN GROUND-MOTION PROCESSING CAPABILITIES

2.1. PACIFIC NORTHWEST SEISMOGRAPH NETWORK

The Pacific Northwest Seismograph Network (PNSN), centered at the University of Washington (UW), operates a network of seismograph stations throughout the Pacific Northwest. The network is operated through a joint effort of the University of Washington and the University of Oregon, and it is funded by the United States Geological Survey (USGS), the United States Department of Energy (USDOE), and the University of Washington. In recent years, the PNSN has developed the capability to generate "ShakeMaps," which are maps of ground-motion intensity. Such maps estimate the earthquake instrumental intensity within the region by interpolating between numerous stations within the network, taking into account geologic conditions. Figure 2.1 shows a ShakeMap generated with measured ground-motions from the 2001 Nisqually earthquake. This particular map shows the estimated acceleration response spectrum ordinate at a period of 0.3 seconds.

As part of this project, the ground-motion processing capabilities at the University of Washington were improved so that the ShakeMaps maps could be made automatically available to WSDOT shortly after an earthquake. This chapter documents improvements to the two main sections of ground-motion processing software: Earthworm (Section 2.2), and ShakeMap (Section 2.3). The programs run on and are connected together by UNIX-like operating systems (Solaris, FreeBSD, and Linux).

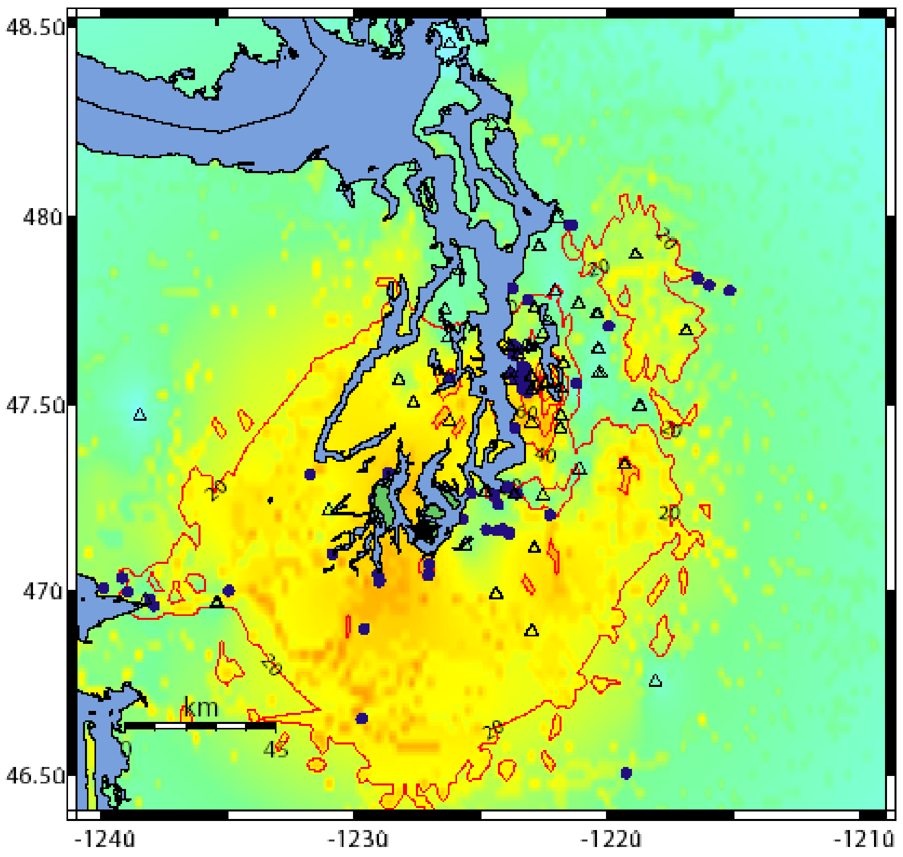

Figure 2.1. ShakeMap showing estimated spectral acceleration at a period of 0.3 sec for the 2001 Nisqually Earthquake (PNSN 2001).

2.2. IMPROVEMENTS TO THE EARTHWORM SOFTWARE

Earthworm is a complicated software suite developed by the community of regional seismic networks in the United States with the support and guidance of the U.S. Geological Survey (USGS 2003). This nationally standardized seismic data recording and exchange system allows data to be collected at local nodes and then broadcast to other nodes. The PNSN operates several local nodes and exchanges data with adjoining networks through Earthworm systems (PNSN 2004).

Seismic waveform trace data from seismographs throughout the Pacific Northwest are received at the PNSN facility at the University of Washington by a variety of means (e.g., internet, microwave, phone). Earthworm aggregates these waveforms onto a common time-base; detects, phase-picks, and locates seismic events; provides an estimate of the size or magnitude of the event; and saves trace data for later manual review by a human seismic analyst. If the estimated magnitude is sufficiently large (M>2.9) and the location is within the authoritative region of the PNSN (Washington state and most of Oregon), Earthworm triggers the transmission of automatic emergency alert messages via e-mail, pager, and FAX to a list of addresses and numbers that includes regional seismologists and emergency managers.

If the event is within the greater Puget Sound area (or M>4.0 in the rest of the Pacific Northwest), Earthworm also triggers automatic processing of ShakeMaps. The basic Earthworm software is extensively documented elsewhere (USGS 2003). However, the special handling of strong-motion data used by ShakeMap was part of this WSDOT project and is documented here.

The basic ShakeMap system was integrated into the automatic, Earthworm processing stream for this project. Figure 2.2 shows the whole Earthworm system, with the parts developed for this project enclosed within the dotted lines (highlighted in green). When an alert event occurs of sufficient size and proximity to trigger ShakeMap, the *alert_prelim* script runs *shake_gen*, which is a shell script that moves files to appropriate directories and runs *gmew_uw*. *Gmew_uw* is a basic Earthworm module, *gmew*, which we have extensively modified to run using data from Earthworm but in UW2 format. Gmew_uw takes the *pickfile* (a list of the phase picks at stations recording the event) and the trace data file (waveforms from the seismographs in UW2 format) and calculates peak ground-motion parameters for the event. These include peak ground velocity, acceleration, and response-spectrum values at periods of 0.3, 1.0, and 3.0 seconds. *Gmew_uw* then outputs this information in XML format for input to the ShakeMap generation process.

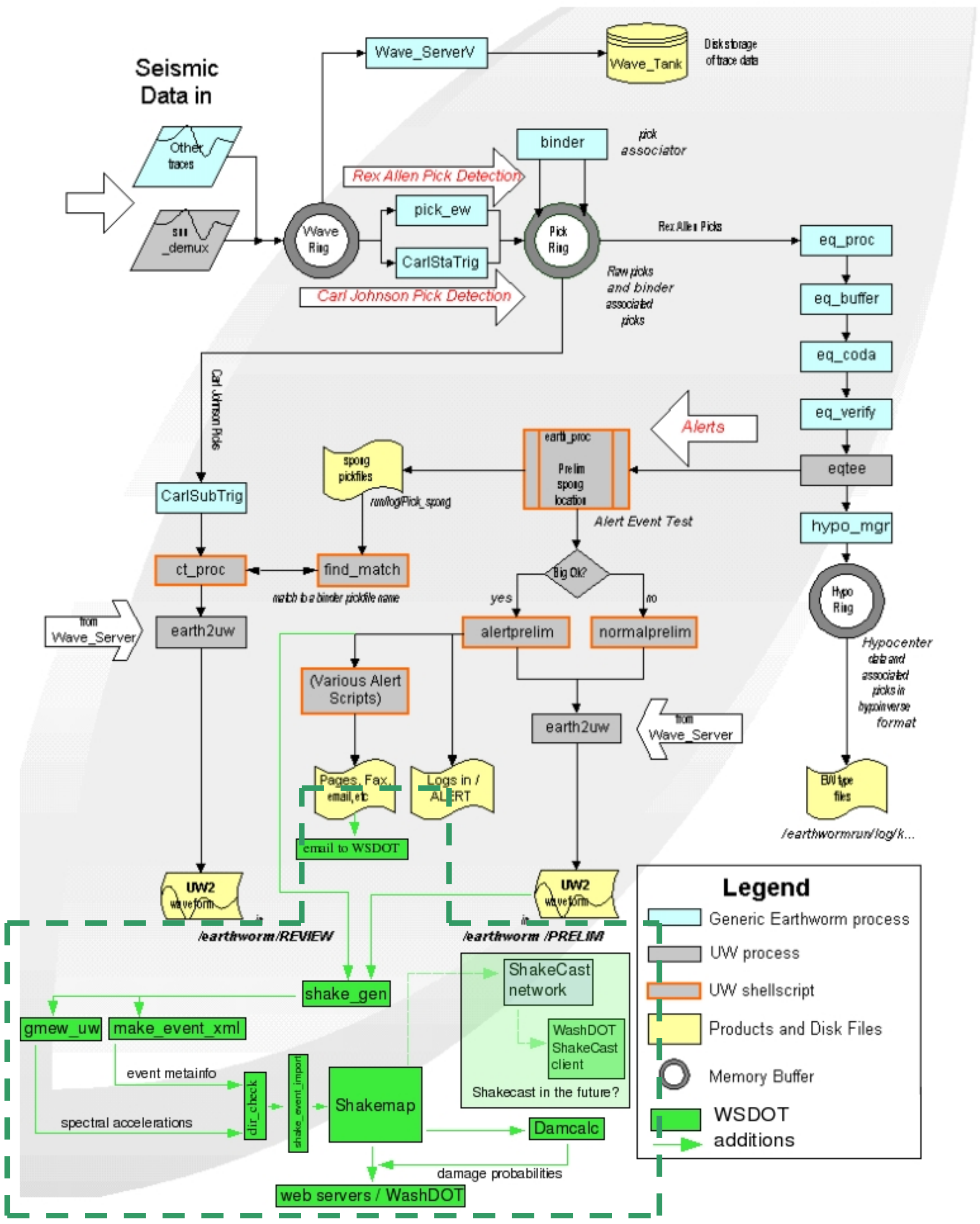

Figure 2.2. UW Earthworm processes operating for the PNSN. The section in the lower left within dotted lines contains the modules written for this project.

Shake_gen also runs a small script, *make_event_xml*, which generates an event parameter XML file that ShakeMap uses for the event location and size estimates. *Shake_gen* then moves these two files to a communal *temp* directory that is mounted by other machines across the network via the Network File System (**NFS)**, where they are accessible to the ShakeMap program.

2.3 IMPROVEMENTS TO THE SHAKEMAP SOFTWARE

ShakeMap, a software package written and maintained by the USGS in the programming language Perl, takes event information and measured peak accelerations, velocities, and spectral estimates measured at seismographs as input, and it produces contour and color-shaded estimates of those values over a region around the event epicenter (Figure 2.1). ShakeMaps include the effect of geology as determined by near surface soils, which can amplify or attenuate strong ground shaking (Wald et al. 1999).

UW researchers updated the regional geologic information to a resolution of 30 seconds (~1.2 km), wrote software that automatically interfaces with Earthworm, modified the ShakeMap configuration files to initiate the bridge damage calculation routine (*dam_calc*; see Chapter 3), and arranged to transfer the resulting data along with standard ShakeMap products to special Web and FTP sites. Outside the scope of this research project, the UW team is exploring the use of ShakeCast (Gatekeeper Systems 2004) for an additional data/information delivery mechanism. ShakeCast has the potential to automatically and robustly deliver maps and tables that include icons for bridges that may have been damaged.

The interface between the Earthworm code and the ShakeMap code is provided by the *dir_check* script, as shown in Figure 2.2. This script runs on each ShakeMap machine and checks the communal directory every few seconds to see whether new files have been written there. When new files appear, having been generated by the Earthworm module *shake_gen*, the *dir_check* script moves them into the input directory for ShakeMap and then starts the master ShakeMap program called *shake*. Because ShakeMap may be running on more than one machine, *dir_check* has two modes, master and slave. In the master mode, it keeps a list of all of the machines that will need the output from *shake_gen*. This mode has to be configured ahead of time and will not remove files from the communal directory unless an unlock file from each machine on its roster is present. The slave version of *shake_gen* gets a copy of the files in the communal directory for its version of ShakeMap and leaves an *unlock* file to communicate with the master. After all of the unlock files are present, the master *dir_check* deletes the data and event meta-info files from the communal directory. The first thing that *shake* does when it runs is get the transferred data files with a script called *shake_event_import*, which moves the XML files with appropriate names into the appropriate directory within the ShakeMap structure.

The standard ShakeMap code is then run to generate image files and an ASCII table of shaking parameters. These parameters serve as input to the *dam_calc* routine, the new routine that estimates the probability of damage for bridges. This new routine is described in Chapter 3.

CHAPTER 3

ESTIMATES OF LIKELIHOOD OF BRIDGE DAMAGE

Damage progression in a bridge during an earthquake is a complex process, which depends on details of the bridge that are not commonly available in bridge databases. For example, the Washington State Bridge Inventory (WSBI) does not provide heights for the columns along the bridge length, nor does it provide reinforcing details. For this reason, UW researchers based the estimated vulnerability of WSDOT bridges on their general characteristics, such as the year of construction.

This chapter describes two procedures that were implemented to estimate the likelihood that a particular bridge would suffer at least slight damage, given the ground-motion intensity indicated by the ShakeMaps (Chapter 2) and certain properties available in the WSBI. The first procedure, developed at the University of Washington, was based on the damage reported for the 2001 Nisqually Earthquake (sections 3.1 and 3.2). The second procedure is contained in the Federal Emergency Management Agency (FEMA) HAZUS software (Section 3.3).

3.1. OBSERVED DAMAGE DURING THE 2001 NISQUALLY EARTHQUAKE

As part of a separate project (mainly funded by the Pacific Earthquake Engineering Research Center, Ranf et al. 2001) UW researchers collected and analyzed bridge damage reported from the 2001 Nisqually earthquake. Because most of the damage to bridges was minor, it was only possible to estimate the likelihood that a bridge might suffer at least slight damage (defined for the purpose of this analysis as any observable damage that was reported by inspectors), rather than to estimate the extent of damage in a particular bridge. The data suggested that a key indicator of likelihood of damage is the year of construction of the bridge, which indirectly accounts for differences in design methodologies and details. Figure 3.1 shows the percentage of bridges damaged as a function of the decade of construction. The percentage of damaged bridges was largest for bridges built before 1941 and smallest for those built after 1970.

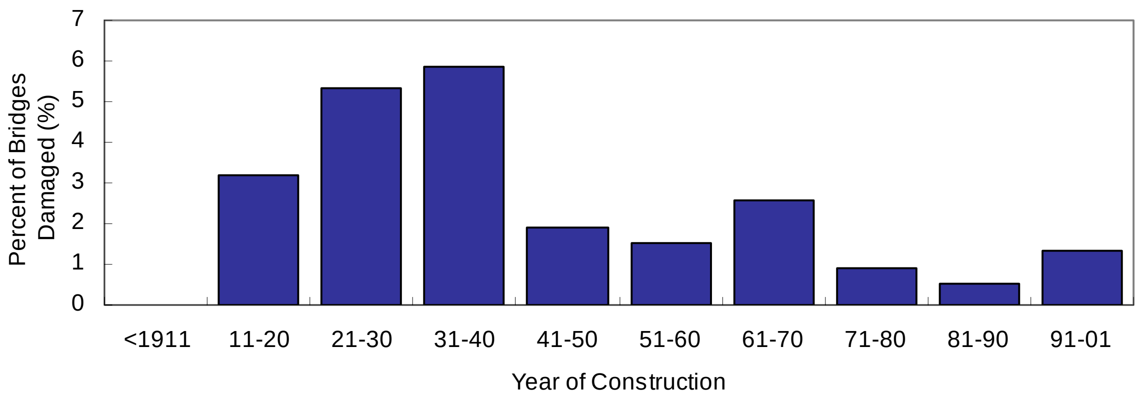

Figure 3.1. Effect of the year of construction on the percentage of damaged bridges.

The estimated spectral acceleration at a period of 0.3 seconds and the bridge type also appeared to influence the likelihood of damage during the Nisqually earthquake. Figure 3.2 shows the likelihood of reported damage as a function of the spectral acceleration, as well as the bridge age and type. Fixed (not moveable) bridges built after 1975 were the least likely to suffer damage, probably as a result of code changes adopted following the 1971 San Fernando earthquake. The fixed bridges that were built before 1941 had higher damage percentages than bridges constructed later. The cause of the large damage percentage for these older bridges is unclear. It is possible that the 1949 Olympia earthquake damaged some of these bridges.

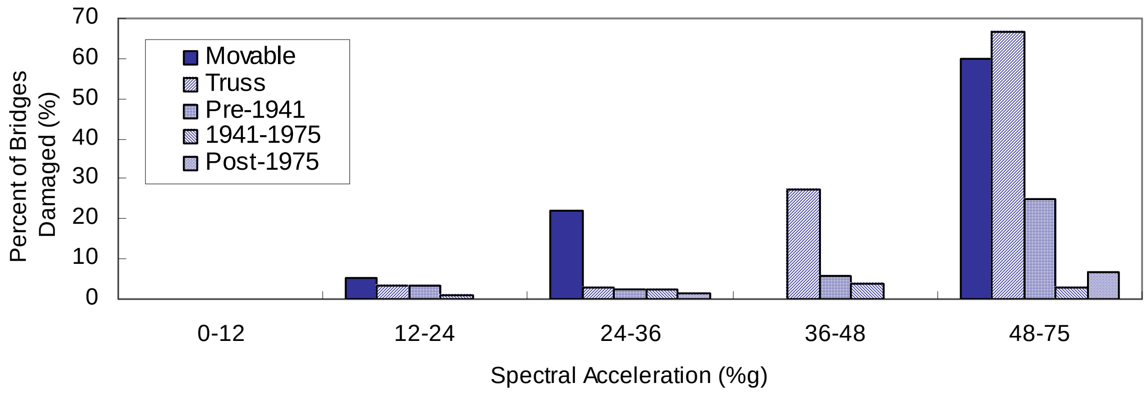

Figure 3.2. Effects of spectral acceleration (T = 0.3 s) and year of construction.

Movable bridges and older truss bridges were particularly likely to suffer damage during the Nisqually earthquake (Figure 3.2). In particular, six of the 43 movable bridges and eight of the 106 trusses within the boundaries of the ShakeMap were damaged, resulting in overall damage percentages of 14 percent for movable bridges and 8 percent for truss bridges. Three of the 15 movable bridges (20 percent) and five of 14 trusses (36 percent) with a spectral acceleration above 0.36g suffered at least slight damage.

3.2. FRAGILITY RELATIONSHIPS BASED ON NISQUALLY EARTHQUAKE DAMAGE

The analysis of earthquake damage indicated that the relationships between earthquake intensity and likelihood of damage (fragility curves) for most bridges in Washington State should be based on the spectral acceleration at a period of 0.3 sec, the year of construction, and whether the bridge is movable or an older steel truss. On the basis of the damage data from the Nisqually earthquake, fragility relationships were developed using a lognormal distribution:

$$P(slight\_damage) = \int_0^{SA(0.3)} \frac{1}{\sqrt{2\pi}\zeta x} \exp\left[-\frac{1}{2} \left(\frac{\ln x - \ln x_m}{\zeta_X} \right)^2 \right] dx \tag{3.1}$$

where *P(slight_damage)* is the bridge damage probability, $S_a(0.3)$ is the estimated spectral acceleration at a period of 0.3 sec (in g), and x_m and ζ are the median and the log-standard deviations of the fragility curves, respectively. On the basis of the Nisqually data, the median spectral accelerations were determined to be

$$x_m = \begin{cases} 0.55 & (pre-1976\ truss) \\ 0.6 & (movable) \\ 0.9 & (pre-1941) \\ 1.4 & (1941-1975) \\ 1.6 & (post-1975) \end{cases}$$

The values for the log-standard deviations of each of the categories (ζ) were set at 0.6 to mirror those used in HAZUS 99. The resulting fragility curves are plotted in Figure 3.3.

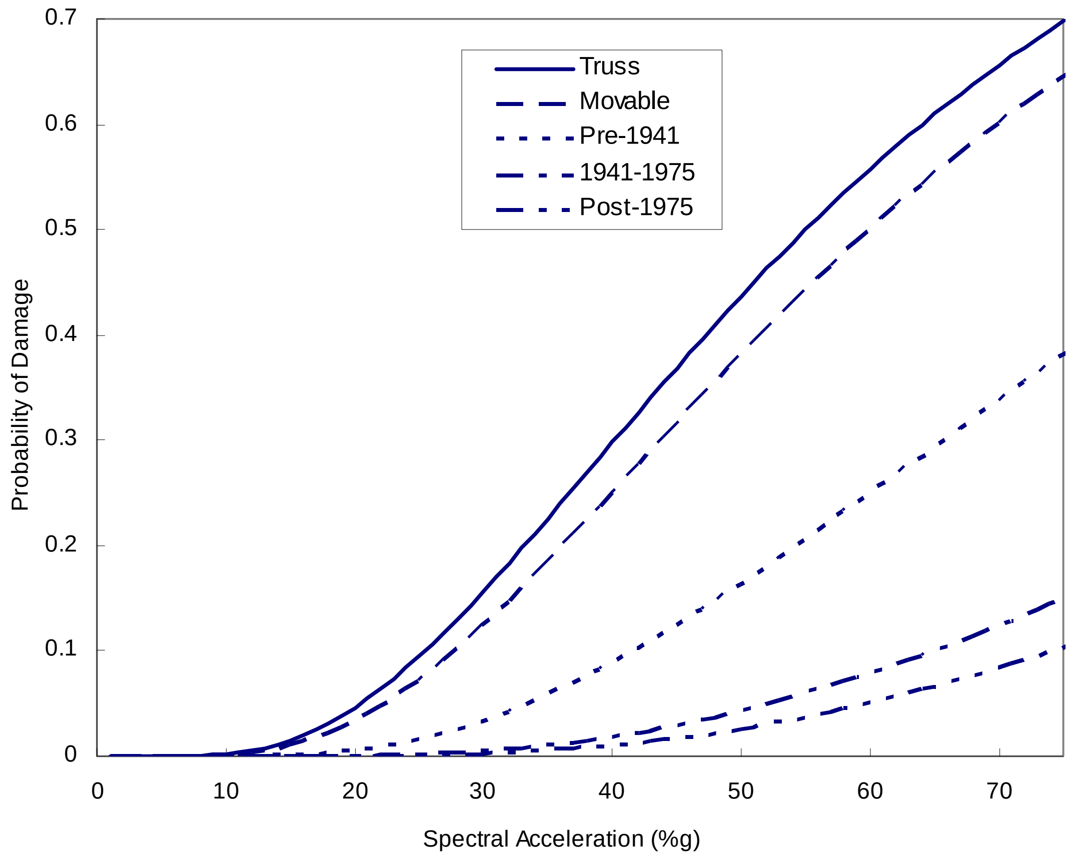

Figure 3.3. Fragility curves for at least slight damage accounting for bridge age and type.

For the Nisqually earthquake, these curves provided more accurate estimates of the damage probability for each of the bridge categories than were provided by the HAZUS relationships (Section 3.3). Table 3.1 illustrates the procedure by means of an example.

Table 3.1. Example implementation of Nisqually-based procedure for Anderson Creek bridge and Nisqually earthquake.

Procedure		Example	
1.	Choose Bridge	Anderson Creek	
		NBI #001706C	
		Year Built:	1933
		Bridge Type:	Reinforced Concrete
			Not Moveable
		Bridge Classification	Pre-1941
3.	Find median and log-standard deviation for the bridge	x_m	0.9
		ζ	0.6
4.	Estimate spectral acceleration at the bridge location (T = 0.3 s)	Sa (T = 0.3 s) =	0.25 g
5.	Calculate the probability that the bridge will have at least slight damage (Eq. 1.1 or Fig. 3.3)	Pd = 1.6%	

As shown in Table 3.1, the Nisqually-based procedure estimated a 1.6 percent probability that the Anderson Creek bridge (NBI #001706C) sustained at least slight damage during the Nisqually earthquake.

3.3. HAZUS 99 FRAGILITY RELATIONSHIPS

As part of this project, a second set of fragility relationships was incorporated into the analysis software. HAZUS 99 (FEMA 1999) provides fragility relationships for several levels of damage, but for this project, only the lowest level of damage was considered in order to facilitate comparison between the HAZUS procedure and the one described in the preceding section. HAZUS classifies bridges on the basis of a variety of characteristics, including age, bridge location, material type, and design type. The HAZUS bridge-classification scheme is summarized in Table 3.2.

Table 3.2. Description of HAZUS bridge classes.

HAZUS Class	NBI Class	State	Year Built	I_{shape}	Design	Description
HWB1	All	Non-CA	< 1990	0	Conventional	Length > 150 m
		CA	< 1975			
HWB2		Non-CA	≥ 1990		Seismic	
		CA	≥ 1975			
HWB3	All	Non-CA	< 1990	1	Conventional	Single Span
		CA	< 1975			
HWB4		Non-CA	≥ 1990		Seismic	
		CA	≥ 1975			
HWB5	101-106	Non-CA	< 1990	0	Conventional	Multiple Column Bent Simple Support Concrete
HWB6		CA	< 1975			
HWB7		Non-CA	≥ 1990		Seismic	
		CA	≥ 1975			
HWB8	205-206	CA	< 1975	0	Conventional	Single Column Bent, Box Girder, Continuous Concrete
HWB9		CA	≥ 1975		Seismic	
HWB10	201-206	Non-CA	< 1990	1	Conventional	Continuous Concrete
		CA	< 1975			
HWB11		Non-CA	≥ 1990		Seismic	
		CA	≥ 1975			
HWB12	301-306	Non-CA	< 1990	0	Conventional	Multiple Column Bent Simple Support, Steel Length More Than 20 m
HWB13		CA	< 1975			
HWB14		Non-CA	≥ 1990		Seismic	Multiple Column Bent Simple Support, Steel
		CA	≥ 1975			
HWB15	402-410	Non-CA	< 1990	1	Conventional	Continuous Steel Length More Than 20 m
		CA	< 1975			
HWB16		Non-CA	≥ 1990		Seismic	Continuous Steel
		CA	≥ 1975			
HWB17	501-506	Non-CA	< 1990	0	Conventional	Multiple Column Bent Simple Support Prestressed Concrete
HWB18		CA	< 1975			
HWB19		Non-CA	≥ 1990		Seismic	
		CA	≥ 1975			
HWB20	605-606	CA	< 1975	0	Conventional	Single Column Bent Box Girder, Continuous Prestressed Concrete
HWB21		CA	≥ 1975		Seismic	
HWB22	601-607	Non-CA	< 1990	1	Conventional	Continuous Concrete
		CA	< 1975			
HWB23		Non-CA	≥ 1990		Seismic	
		CA	≥ 1975			
HWB24	301-306	Non-CA	< 1990	0	Conventional	Multiple Column Bent Simple Support, Steel
HWB25		CA	< 1975			
HWB26	402-410	Non-CA	< 1990	1		Continuous Steel
HWB27		CA	< 1975			
HWB28						All Other Bridges

In HAZUS 99, the definition of slight damage for each of the bridges is the same: minor cracking and spalling on the abutment or hinges, minor spalling of the columns (only cosmetic repair needed), and minor cracking on the deck (FEMA 1999).

The probability of a bridge sustaining slight damage during an earthquake is calculated by using a lognormal cumulative density function

$$P(slight\_damage) = \int_0^{SA(1.0)} \frac{1}{\sqrt{2\pi}\zeta x} \exp\left[-\frac{1}{2}\left(\frac{\ln x - \ln x_m}{\zeta_X}\right)^2\right] dx \qquad (3.2)$$

where P(*slight_damage*) is the probability that the bridge will experience at least slight damage, SA_X is the spectral acceleration at the location of that bridge, and x_m and ζ are the median and log-standard deviation of the damage data.

For each bridge, the log-standard deviation was assumed to be 0.6 for ground shaking. The standard medians spectral accelerations (T = 1.0 s), $x_{m,old}$, for each bridge type are listed in Table 3.3.

Table 3.3. Median spectral accelerations for each HAZUS bridge class.

HAZUS Class	$x_{m,old}$	HAZUS Class	$x_{m,old}$	HAZUS Class	$x_{m,old}$	HAZUS Class	$x_{m,old}$
HWB1	0.40	HWB8	0.35	HWB15	0.75	HWB22	0.60
HWB2	0.60	HWB9	0.60	HWB16	0.90	HWB23	0.90
HWB3	0.80	HWB10	0.60	HWB17	0.25	HWB24	0.25
HWB4	0.80	HWB11	0.90	HWB18	0.30	HWB25	0.30
HWB5	0.25	HWB12	0.25	HWB19	0.50	HWB26	0.75
HWB6	0.30	HWB13	0.30	HWB20	0.35	HWB27	0.75
HWB7	0.50	HWB14	0.50	HWB21	0.60	HWB28	0.80

The standard median spectral accelerations listed in Table 3.3, $x_{m,old}$, for each HAZUS bridge class were adjusted to the bridge specific median spectral acceleration, $x_{m,new}$, by using Equation 3.3.

$$x_{m,new} = x_{m,old} \times F \tag{3.3}$$

where F is a modifying factor, given by the equation

$$F = \begin{cases} 1 & I_{shape} = 1 \\ \min(1, K_{shape}) & I_{shape} = 0 \end{cases} \tag{3.4}$$

where I_{shape} is given in Table 3.2 for each bridge class, and K_{shape} is determined by the equation

$$K_{shape} = 2.5 \times \frac{SA(T = 1.0s)}{SA(T = 0.3s)} \tag{3.5}$$

Table 3.4 outlines the procedure for calculating the probability that a bridge in the WSDOT database has sustained at least slight damage, given the information in tables 3.2 and 3.3, and given the estimated spectral accelerations at the bridge location from the Nisqually earthquake. The probability of slight damage is illustrated for an example bridge.

As shown in Table 3.4, the HAZUS procedure estimated a 13 percent probability that the Anderson Creek bridge (NBI #001706C) sustained at least slight damage during the Nisqually earthquake. This probability is approximately eight times larger than the probability resulting from implementation of the Nisqually-based procedure (Section 3.2)

Table 3.4. Example Implementation of HAZUS 99 Procedure for Anderson Creek Bridge and Nisqually Earthquake.

Procedure		Example	
1.	Choose Bridge	Anderson Creek	
		NBI #001706C	
2.	Choose HAZUS class for the bridge	WSDOT Bridge	
		Year Built:	1933
		Max Span Length:	40 ft
		Main Span Material:	Reinforced Concrete
		Main Span Design:	Simple Support
			Multi-Column Bent
		HAZUS Class (Table 1)	HWB5
3.	Find median and log-standard deviation for the bridge	Xm,old (Table 2) =	0.25
		ζ =	0.6
4.	Estimate the spectral acceleration at the bridge location (T = 0.3 s and 1.0 s)	SA (T = 0.3 s) =	0.25 g
		SA (T = 1.0 s) =	0.13 g
5.	Calculate bridge specific median spectral acceleration (T = 1.0 s)	Ishape (Table 1) =	0
		F (Eq. 3) =	min(1,Kshape)
		Kshape (Eq. 4) =	2.5*0.13/0.25
			1.3
		F =	1
		Xm,new (Eq. 2) =	0.25
6.	Calculate the probability that the bridge will have at least slight damage	$P(slight\_damage) = \int_0^{0.13} \frac{1}{\sqrt{2\pi}\,0.60x} \exp\left[-\frac{1}{2}\left(\frac{\ln x - \ln[0.25]}{0.60}\right)^2\right] dx$ $P(slight\_damage) = 13\%$	

3.4. IMPLEMENTATION OF DAMAGE PROBABILITY ALGORITHMS

This section describes a new module for ShakeMap, called *dam_calc*. This module was written to combine the WSDOT bridge inventory data and shaking estimates from ShakeMap (Chapter 2) to provide WSDOT with a prioritized list of bridges that may have been damaged during an earthquake. The list will contain damage probabilities calculated for each bridge in the state bridge inventory with a method based on Nisqually earthquake damage (Section 3.2) and damage probabilities calculated by the HAZUS 99 procedure (Section 3.3). The software (*dam_calc*) that calculates these damage probabilities and produces this list was written in Perl to be compatible with the other ShakeMap modules. *Dam_calc* is executed by ShakeMap, as configured in the *shake.conf* configuration file and its output integrates with the ShakeMap transfer method as configured in the *transfer.conf* file. The *Dam_calc* code is provided in Appendix A.

Dam_calc first reads in the bridge inventory file and the *grid.xyz* file created by ShakeMap. The *grid.xyz* file contains a list of regularly spaced geographic points and values of peak ground velocity, peak ground acceleration, and spectral accelerations at periods of 0.3, 1.0, and 3.0 seconds at those points. The bridge inventory file is a subset of the WSBI database file and needs to be provided by WSDOT, as discussed in Section 3.5 .

After the inventory and grid files have been read, *dam_calc* determines which grid points the first bridge lies between by using a recursive algorithm that quickly searches through the available grid points. *Dam_calc* will also successfully find the appropriate points if the bridge is coincident with a grid point, exactly between two grid points, or on the edge of the ShakeMap. *Dam_calc* then calculates a weighted average of the peak ground motion and spectral parameters at the location of the bridge. It uses a standard weighted average, the weights falling off proportionally to $1/d^2$, where d is the distance from the bridge to the grid point. The exponent in the denominator of the weight expression can be adjusted by changing a parameter listed at the top of *dam_calc.pl*.

After the effective peak ground motion and spectral parameters at the point of the bridge have been calculated, the UW (Section 3.2) and HAZUS (Section 3.3) methods are applied to calculate the probabilities of damage. These values are then output with identifying information from the bridge inventory (e.g., longitude, latitude, Bridge ID #), with the damage probabilities sorted in descending order by the external standard UNIX program, *sort*.

<u>3.5. INPUT FILES FOR DAMAGE PROBABILITY PROGRAM</u>

The Washington State Bridge Inventory is stored in an Access database. For this project, the input file to *dam_calc* was an Excel-compatible spreadsheet containing only selected information from the full database. The columns included in the spreadsheet are listed in Table 3.5.

Table 3.5. Columns included in the Excel input file

Column #	Information Contained in Each Column
1	Latitude
2	Longitude
3	Year
4	Bridge Number
5	Span Type (NBI Classification)
6	Bridge Name
7	Main Span Material
8	DOT Bridge ID
9	Length
10	NBI Length
11	Max Span

Scripts were written to convert the spreadsheet format into the input format for *dam_calc*. This script selects certain columns of data, merges them with colons between the columns, and makes sure that the end-of-line characters are correct. The procedure that is used to generate the bridge inventory input file for *dam_calc* is the following:

1. Export the spreadsheet file to a colon-separated file.
2. Pass the colon separated file to *inv_trunc.pl* and its output into the file, $SHAKE_HOME/lib/*dam_calc* which is the input file for *dam_calc*.

This procedure need only be done when the bridge inventory information changes enough to warrant preparing a new input file, for example, on a yearly basis. This input file needs to be generated by WSDOT (which has access to the updated database) and transmitted to PNSN staff.

CHAPTER 4

TRANSMISSION OF INFORMATION TO WSDOT

4.1. TRANSMISSION PROCESS

In the event of an "alert" (usually triggered by an earthquake of magnitude 4.0 or greater), WSDOT will automatically receive an e-mail message containing basic earthquake parameters and instructions for where to find ShakeMap information. The e-mail will be addressed to *quake@wsdot.wa.gov* and contain the subject line "Preliminary Alert Earthquake." WSDOT will automatically distribute this message internally to the WSDOT Disaster Advisory Group, a statewide distribution.

The e-mail alert will be followed approximately 10 to 15 minutes later by a ShakeMap and supplementary bridge damage probability data, which will be placed in Web and FTP areas. Copies of the ShakeMap files will be pushed to the WSDOT FTP server (in the */Public/ShakeMaps/shake/<eventID>* directory), along with the file named <eventID>_dam_prob.uw, which will contain the bridge damage probability information (by both the UW and HAZUS methods). If additional strong-motion data become available (e.g., from dial-up stations), the shakemap process will be repeated, and the updated information will be transmitted to WSDOT.

The ShakeMap files sent to WSDOT that will be generated by Version 3.0 of ShakeMap should not be substantially different from the publicly accessible ones (at www.pnsn.org/shake). A minor difference is that the WSDOT maps will not include shaded topography but instead will include the outlines of major highways. To view the ShakeMaps that are in the WSDOT FTP directory, engineers can use the Web browser address *ftp://ftp.wsdot.wa.gov//Public/ShakeMaps/shake/<eventID>* (where <eventID> is the date-time identifying label for this earthquake*)* or copy the <eventID> directory to a directory on their own computers and then simply open the *intensity.html* file. This file will contain links to the various maps and products for the event. Note that the damage probability files will not be accessible directly from this page because they will not be part of standard ShakeMap distributions. Rather than opening the *intensity.html* file in the event directory, viewers can point their browsers at the *<eventID>_dam_prob.hazu*s file. This file could be transferred to the viewers' computers and opened with a text reading program or editor, or it could be read into a spreadsheet or database program.

To provide system redundancy, copies of all of these files will also be available on a private, high-availability Web server with the address of *http://grasso.ess.washington.edu/shake*. These files can be accessed from any location with Internet access, such as the state emergency management center. If the Internet were completely disabled, the bridge lists could be transmitted by Fax or courier.

The FTP server at WSDOT has account/password protection. This protection is configured in the ShakeMap 'pw' directory. The account on the FTP server must have delete/rename access on the directories into which it places files. This is because of a file push sanity checking mechanism in ShakeMap that renames files once they have been placed in a temporary fashion. The push fails without that permission. Also, on the ShakeMap machine, passive FTP must be enabled to get around any firewalls. (A strong firewall is installed on the DOT ShakeMap machine at UW.) This is accomplished by setting the environment variable in the environment from which ShakeMap is run 'FTP_PASSIVE' to 1.

4.2. FILE FORMAT FOR PRIORITIZED LISTS OF BRIDGES

This section provides details of the format of the bridge damage output file. Full documentation of the ShakeMaps, which include maps of peak acceleration and spectral acceleration, are available on the PNSN website.

The dam_calc output file will be named *<eventID>_dam_prob.uw.* , where *<eventID>* is the date-time of the earthquake. This file will be generated with each seismic event of magnitude 4.0 or greater. The output file will contain lines (ended by a 0x0A character) that have seven comma-separated fields. The first line will be a header containing brief descriptions of the fields. The character format will be exactly the same. The lines will be sorted in descending order by the probability of damage. This means that the first data line will contain the bridge most likely to be damaged. Note that the first column will contain the damage probability calculated by the UW method, and the second column will contain the probability calculated by the HAZUS method. The default sort will be on the UW method. Table 4.1 delineates these fields; Table 4.2 provides sample output.

It is possible that a future event will occur in a location where the ShakeMap will not generate any data that overlap geographically with the Washington State Bridge Inventory. For example, if an event were located in Oregon, the ShakeMap produced would center around the event epicenter. If the boundaries of the ShakeMap created did not include any place where a bridge in the inventory was located, there would be no output from *dam_calc*, and the list files of damage estimates would be empty.

Table 4.1 File format of prioritized lists of bridges

	Field 1	Field 2	Field 3	Field 4	Field 5	Field 6	Field 7	Field 8
Name	UW_Pd (Probability of damage)	HAZUS_Pd (Probability of damage	Estimated Spectral Accelera-tion at 0.3 sec,	DOT ID	Bridge Name	Bridge Number	Bridge Latitude	Bridge Longitude
Specifi-cation	Floating point, characters: 1.5	Floating point, characters: 1.5	Floating point, characters: 1.5	Text String (arbitrary length)	Text String (arbitrary length)	Text String (arbitrary length)	Floating point, characters: (-)*.5	Floating point, characters: (-)*.5
Origin	Calculated by dam_calc	Calculated by dam_calc	Calculated by dam_calc	Directly from the bridge inventory	Directly from the bridge inventory	Directly from the bridge inventory	From bridge inventory, floating point format adjusted by dam_calc.	From bridge inventory, floating point format adjusted by dam_calc.

Table 4.2 Sample output for Nisqually event

```
UW_Pd, HAZUS_Pd, br_psa03, DOTID,  BRName, BRNum, BRLat, BRLon

0.62717,0.02745,66.82,"08109700","42ND AVENUE SOUTH BR","0000TUK14",47.49167,-122.26667

0.62357,0.07761,72.48,"08433700","SOUTH PARK BRIDGE","     3179",47.52978,-122.31408

0.52094,0.05904,61.92,"0014459A","DUWAMISH RIVER"," 99/530W",47.54167,-122.33167

0.52094,0.05904,61.92,"0004872A","DUWAMISH R B"," 99/530E",47.53262,-122.33485

0.36877,0.12327,44.98,"000000JD","PUYALLUP R","162/6",47.13167,-122.23333

0.33255,0.02641,42.42,"08329400","ALVORD ""T""","     3130",47.37167,-122.23000

0.32941,0.08021,42.20,"08541900","STUCK RIVER","SUM24204A",47.20333,-122.24500

0.30967,0.06807,40.82,"0003960A","PUYALLUP R","167/20E",47.20333,-122.29333
```

ShakeMaps would be transmitted to WSDOT even if the ShakeMap did not overlap with any bridges in the WSBI. This unusual situation (of empty lists) could be identified easily by viewing the ShakeMap image, which would show only regions in the Pacific Northwest that did not have WSDOT bridges. Log files, which would not be pushed to the WSDOT website, could also be checked (by contacting PNSN) to verify that the program did indeed run.

REFERENCES

Earthquake Engineering Research Institute (EERI), *The Nisqually Earthquake of 28 February 2001: Preliminary Reconnaissance Report*, Earthquake Eng. Research Institute. Oakland, CA, 2001.

Federal Emergency Management Agency (FEMA), “Direct Physical Damage to Transportation Systems,” Chapter 7, in HAZUS 99 Service Release 1 (SR1) Technical Manual, Washington, D.C., 1999.

Gatekeeper Systems, “Shakecast Development Information Website,” http://www.shakecast.org/, 2004.

Pacific Northwest Seismograph Network (PNSN). “Home Page,” University of Washington, http://www.geophys.washington.edu/SEIS/PNSN/, 2004.

Pacific Northwest Seismograph Network (PNSN). “Shake Map,” University of Washington, http://www.geophys.washington.edu/shake/0102281854/intensity.html , 2001

Ranf, R.T., Eberhard, M.O. and Berry, M.P., "Damage to Bridges During the 2001 Nisqually Earthquake," Pacific Earthquake Engineering Research Center Report PEER-2001-15, University of California, Berkeley, November, 2001.

United States Geological Survey (USGS), “Earthworm Documentation, http://folkworm.ceri.memphis.edu/ew-doc/, 2003.

Wald, D.J., Quitoriano,Vincent, Heaton, T.H., Kanamori, H., Scrivner, C.W., and Worden, C. Bruce, TriNet “ShakeMaps": Rapid Generation of Instrumental Ground Motion and Intensity Maps for Earthquakes in Southern California, *Earthquake Spectra*, 15, 537-556, 1999.

APPENDIX A

COMPUTER CODE FOR DAMAGE PROBABILITY CALCULATION

NOTE: The following listing of the PERL code contains long lines that have wrapped in the listing, and thus care should be taken in using this listing for anything other than manually reviewing the code. For a machine readable copy of the code, contact seisops@ess.washington.edu.

```
#!/usr/local/bin/perl

#********** dam_calc - the bridge damage probability calculator               ****
#********** by: Jay LaBelle (2004)

#***** General Program documentation *************************************************
# USAGE:
#  The program can be run in 2 ways.  One way is in standalone mode, where dam_calc
# will simply analyze the bridge inventory it is fed against the grid file that it is
# fed.  The command line parameters for this mode are:
# ./dam_calc <bridge_inventory_file> <grid_file>
#  In this mode, two files will be produced in the directory from which the program
# was executed. (dam_prob.hazus and dam_prob.uw).  These files contain the damage
# probabilities calculated in the two different manners.
#  The second way to run dam_calc is through ShakeMap.  In this mode, dam_calc should
# be added to the ShakeMap execution list in the 'shake.conf' file in the /config
# directory of the ShakeMap tree.  The 'nodep' ShakeMap option should be used for
# executing dam_calc.  ShakeMap will then execute dam_calc passing the '-event' flag
# and the event ID.  To use this mode, the file locations and paths need to be
# configured later on in this script. (Text search for 'FILENAMES/PATHS' to see where)
#
# REQUIREMENTS:
# - dam_calc needs Perl, obviously.
# - It also requires the Perl module 'Math::CDF' to
# be accessible through the normal Perl means.  (The interpretor needs to be able to
# find it.)
# - dam_calc needs the UNIX 'sort' command.  The path and flags for 'sort' need to be
# configured in this script also.  (Text search for 'SORT CONFIG' in this script to
# find it.)
# - inventory file: This is the Washington State Bridge Inventory.  (Really any
# inventory file would work as long as it has the expected format.)
# - dam_calc needs a grid file from ShakeMap.
#
# OUTPUT:
# dam_calc produces either one or two files for each run, labelled differently depending on which
# mode produced them.  If in standalone mode, ./dam_prob.hazus and ./dam_prob.uw will
# be created.  If in 'ShakeMap' mode, then the files created will be called something
# based on the event ID of the event, and will be placed in the locations specified
# by the path variables spoken about before. By default only one file is produced containing both UW
#and HAZUS output results.
#***************************************************************************************

#***************************************************************************************
```

```
#**** FUNCTION DOCUMENTATION  **********************************************
#******************************************************************************

#***** calc_accels() *********************************************************************
# USAGE:
#   ($br_psa03, $br_psa10, $br_pga, $gr_lat, $gr_lon) = calc_accels($br_ref);
#
# PARAMETERS: $br_ref is a reference (like a pointer) to a bridge array.  This bridge
#   array contains all of the information about the bridge.
#
# RETURN VALUES: $br_psa03, $br_psa10, $br_pga are psa03. psa10, and pga for the
#   bridge.  These values aren't updated in the bridge array by this function.
#   It is up to the calling function to change the bridge array.
#   $gr_lat, $gr_lon are the latitude and longitude of the last point used in the
#   calculation.  These values don't make sense any more, but are still returned
#   because they were used in development and haven't been removed yet.
#
# DESCRIPTION:
# Calculates the ground motion parameters at the location of the specific bridge
# passed to it.  If the bridge lies between 4 points, (inside the quadrilateral
# limited by 4 grid points,) it uses all 4 points to calculate a weighted average
# of the ground motion parameters at the location of the bridge.  The falloff of the
# weighting is described by the coefficient in the "CONFIGURABLE CONSTANTS" section
# below.  If the bridge lies directly between 2 points, or is on the edge of the
# ShakeMap area, it uses only 2 points to calculate the average.  If it is exactly
# coincident with a point, it uses only that point and no average is calculated.
# This function calls other functions to perform these calculations.
# *******************************************************************************************

#***** find_grids() ***********************************************************************
# USAGE:
#   @boundpts = find_grids($bridge_ref);
# PARAMETERS:
#   $br_ref is a reference (like a pointer) to a bridge array.  This bridge
# array contains all of the information about the bridge.
# RETURN VALUES:
#   @boundpts = ($sw, $nw, $se, $ne)
# DESCRIPTION:
#   This function uses 2 recursive algorithms to find the grid points that surround
# the bridge.  We know that the grid points are sorted in order of latitude and then
# longitude.  That is to say that latitude takes precedence, and then for all points of
# the same latitude, the points are sorted by longitude.  This enables us to search
# first for the points whose latitude is immedately adjacent to the bridge, and then
# search for the longitude points amongst this set.  This is how this function works.
# The function divides the gridpoint set in half by latitude, and then calls itself
# using the half that contains the desired points.  In this fashion, the algorithm
# execution time scales proportionally to the base-2 log of the total number of grid
# points.  The process is then repeated using the longitude.  The result is 1,2, or 4
# grid points that surround the bridge.
#
#   Note: If nw or ne == 0, then the bridge matches the latitude of grid point exactly.
# If se and sw are the same, or ne and nw are the same, then the bridge matches the
# longitude of a grid point exactly.  If nw or ne == 0 and se and sw are the same,
# then the bridge is on a grid point.
#
#****************************************************************************************
```

```
#***** lat_search() ********************************************************************
# USAGE (example):
#   ($beg, $end) = lat_search($beg, $end, $bridge);
# PARAMETERS:
#   $beg and $end are the beginning and ending indices of the segment of the grid point
# array to be searched.  $bridge is the reference to the bridge in question.
# RETURN VALUES:
#   $beg and $end are the beginning and ending indices of the segment of the grid point
# array that contains the set of all grid points at the latitudes immediately adjacent
# to the bridge.  (The row above and below, or the row the bridge is on.)
# DESCRIPTION:
#   Uses a recursive algorithm to find the grid points at the latitudes immediately
# adjacent to the bridge.  (The row above and below, or the row the bridge is on.)
#***************************************************************************************

#***** lon_search() ********************************************************************
# USAGE (example):
#   ($sw_pt, $se_pt) = lon_search($beg, $end, $bridge);
# PARAMETERS:
#   $beg and $end are the beginning and ending indices of the segment of the grid point
# array to be searched.  $bridge is the reference to the bridge in question.
# RETURN VALUES:
#   $beg and $end are the beginning and ending indices of the segment of the grid point
# array that contains the set of all grid points at the longitudes immediately adjacent
# to the bridge.  (The column east and west, or the column the bridge is on.)
# DESCRIPTION:
#    Uses a recursive algorithm to find the grid points at the longitudes immediately
# adjacent to the bridge.  (The column east and west, or the column the bridge is on.)
# Since this is usually called with the subset of grid points that are immediately
# north and south of the bridge, this effectively returns indices that are either the
# same or differing by 1.  This is called twice in most cases, once on the northern row
# and once on the southern row, to give up to 4 points total.
#***************************************************************************************

#***** usage() *************************************************************************
# USAGE:
#   usage()
# PARAMETERS:
#   none.
# RETURN VALUES:
#   none.
# DESCRIPTION:
#   Prints out usage info if the command line parameters don't make sense.
#***************************************************************************************

#***** parse_gridfile() ****************************************************************
# USAGE:
#   parse_gridfile();
# PARAMETERS:
#   none
# RETURN VALUES:
#   none
# DESCRIPTION:
#   Reads the grid file, sets the global variables that are associated with the
# metainfo in the grid file, and loads the grid point arrays.  Also checks to make sure
# that the grid point array is loaded with good data.  Keeps a count of number of grid
# points read. (global var)
#***********************************************************************************

#***** parse_inventory() ************************************************************
# USAGE:
#   parse_inventory();
# PARAMETERS:
#   none
# RETURN VALUES:
#   none
# DESCRIPTION:
#   Reads the bridge inventory file, loads the bridge arrays.  Checks to make sure
# that the bridge data is good.  Keeps a count of number of bridges read. (global var)
#********************************************************************************

#***** UW_prob_calc() ***************************************************************
# USAGE:
#   $UWPd = UW_prob_calc($br_year, $br_span, $br_psa03);
# PARAMETERS:
#   $br_year, $br_span, $br_psa03 are all bridge parameters that are calculated or
# taken directly from the inventory.  They are all stored in the bridge array, but
# for the sake of this example, (and clarity elsewhere in the code,) have been
# renamed/equated.
# RETURN VALUES:
#   $UWPd is the probability of this bridge being damaged.  It is a floating point
# value between 0 and 1.
# DESCRIPTION:
#   This function calculates the probability of bridge damage per the method shown
# in the paper by Ranf and Eberhard.
#**********************************************************************************

#***** HAZUS_prob_calc() ************************************************************
# USAGE:
#   $HAZUSPd = HAZUS_prob_calc($br_ref, $br_psa03, $br_psa10, $br_pga);
# PARAMETERS:
#   $br_ref, $br_psa03, $br_psa10, $br_pga are all bridge parameters that are calculated
# or taken directly from the inventory.  They are all stored in the bridge array, but
# for the sake of this example, (and clarity elsewhere in the code,) have been
# renamed/equated.  $br_ref is the reference to the bridge array in question, $br_psa03,
# $br_psa10, are spectral accelerations, and $br_pga is the peak ground acceleration of
# the bridge in question.
# RETURN VALUES:
#   $HAZUSPd is the probability of this bridge being damaged calculated.
# This value is a floating point value between 0 and 1.
# DESCRIPTION:
#   This method is described in the HAZUS documentation and a comparison with
# the UW method is in the paper by Ranf and Eberhard.
#********************************************************************************

#***** HAZUS_classify() *************************************************************
# USAGE:
#   $br_htype = HAZUS_classify($br_ref, $br_length);
# PARAMETERS:
#   $br_ref is the reference to the bridge array in question, and $br_length is a value
# used by the HAZUS classification system that is the NBI length if it exists, or the
# length paramater from the Washington State Bridge Inventory.
# RETURN VALUES:
#   $br_htype is the HAZUS bridge type.  (Integer)
# DESCRIPTION:
#   Classifies the bridge per the standard HAZUS bridge classification method.  Does
# not store the classification in the bridge array.  This is handled by the calling
# function.  This was derived from R.T. Ranf's code.
#
#**************************************************************************************

#**************************************************************************************
#***** DATA/VARIABLE DOCUMENTATION ********************************************
#**************************************************************************************

#***** BRIDGE ARRAY INDICES DOCUMENTATION:
# The bridge data is stored in the following manner: Each bridge has it's info stored
# in an array of the type described by the indices below.  There is another array that
# contains references (like pointers in C) to each of these arrays.  All of these
# arrays are global.  This was done because nearly every function needs them anyway.
#
# *** What all the array indices point to:
#
#0  : Internal index
#1  : Latitude
#2  : Longitude
#3  : Year
#4  : Bridge Number
#5  : Span Type (NBI class)
#6  : Bridge Name
#7  : Material
#8  : DOT Bridge ID
#9  : Length
#10 : NBILength
#11 : Max Span
#12 : Unused
#13 : Unused
#14 : HAZUS bridge classification
#15 : psa03
#16 : grid point latitude (only the last point considered in the cell that contains the bridge -- legacy)
#17 : grid point longitude (see above comment)
#18 : peak ground acceleration (PGA)
#19 : peak spectral accel. (1 Hz) (psa10)
#20 : UW damage probability
#21 : HAZUS damage probability
#********************************************************

#***** GRID POINT ARRAY DOCUMENTATION: What all the array indices point to.
#
#0  : Latitude
#1  : Longitude
#2  : PGA
#3  : Values unused in this program.
#4  : Values unused in this program.
#5  : PSA03
#6  : PSA10
#7  : PSA30
```

```
#*******************************************************************

#*****************************************************************
#***** BEGINNING OF THE ACTUAL CODE ****************************
#*****************************************************************

use strict;
require Math::CDF;

# ****************************************************
# CONFIGURABLE CONSTANTS.  These shouldn't have to be changed, with the possible
# exception of the $sort_command variable.
# NOTE: There are more configurable options farther down in this file.  They
# aren't really constants in the traditional sense though, so they are separate.
# ****************************************************

   # *****Configurable option******************************
   # $sort_command contains the command to which an unsorted bridge damage probability
   # list is piped.  This can be changed, but this should work for most FreeBSD or Linux
   # systems.  The '-r' inverts the sort order, so that the highest probability is listed
   # first.
   # ****************************************************

   # ***** SORT CONFIG *****

   our $sort_command = '/usr/bin/sort -r';

   #Bridge type curve coefficients for the UW method.  See UW method docs for more
   #information.
   #
   # These should not be changed unless you know what you are doing.  These constants
   # directly effect how the damage probability is calculated.
   #
   # These variables are set as they are to make the algorithm appear more similar
   # to the algorithm described by R.T. Ranf in his Matlab code.

   our @uw_coefficients = (90, 0.60, 140, 0.6, 160, 0.6, 60, 0.6, 55., 0.6);
   our $lam1 = $uw_coefficients[0];
   our $xi1 = $uw_coefficients[1];
   our $lam2 = $uw_coefficients[2];
   our $xi2 = $uw_coefficients[3];
   our $lam3 = $uw_coefficients[4];
   our $xi3 = $uw_coefficients[5];
   our $lam4 = $uw_coefficients[6];
   our $xi4 = $uw_coefficients[7];
   our $lam5 = $uw_coefficients[8];
   our $xi5 = $uw_coefficients[9];
   our @uw_years = (1940, 1975);   #separation years, these are the years that separate
                              #the eras in the UW classification scheme.

   our $INTERP_POWER = -2;        #Power to which the distance between bridge location
                              #and grid point is raised to calculate a weighting
                              #for the weighted average.  (-2 implies an inverse
                              #square relationship.)

   our $HAZUS_YEAR = 1975;        #Parameters used in the HAZUS algorithm.  (See HAZUS
                        #docs for more information.)
  our $HAZUS_SD = 0.6;        #HAZUS standard deviation

#*******************************************************************************
#End Constants
#*******************************************************************************

#*****GLOBAL DECLARATIONS
# Includes the arrays that hold either values or references for the bridges.
#*****

#array of references to bridge arrays

our @bridges = ();

# array of references to grid arrays

our @shake_pts = ();

#counters of various things

our $in_count = 0;
our $out_count = 0;
our $grids_in = 0;
our $grids_used = 0;

#mode of operation, from command line

our $method = '';

#The array that holds the metaline data from the header of the gridfile

our @map_params = ();

#filenames/paths -- defined further down in the script.
#The definitions are user configurable, and should be configured appropriately.

our $br_inv_file = '';
our $grid_file = '';
our $uw_out_file = '';
our $hazus_out_file = '';

#event ID -- from command line if 'SM' method is used.

our $evid = '';

#File/IO handles

our $INV_HDL = 0;
our $GRID_HDL = 0;
our $UW_OUT = 0;
our $HAZUS_OUT = 0;

#Boundry lines from the grid file

our $west_bdry = 0;
our $east_bdry = 0;
our $north_bdry = 0;
our $south_bdry = 0;

#Counter for number of bridges processed.

our $processed = 0;
#*****************************************

#***************************************************************************
# Main part of the program.  Calls the other functions and loops.
#***************************************************************************

#***************************
# Parse the command line parameters (this section can be optimized)
#***************************

if ($ARGV[0] eq '-event') {    #being run from ShakeMap
  if (scalar(@ARGV) != 2) {
    usage();
    die();
  }
  else {
    $method = 'SM';         #ShakeMap
    $evid = $ARGV[1];
  }
}
else {                         #being run standalone
  if (scalar(@ARGV) != 2) {
  usage();
  die();
  }
  else {
    $method = 'SA';         #Standalone
  }
}

#***********************
#*** FILENAMES/PATHS: these can be changed by the user as needed.
#***********************

if ($method eq 'SM') {
  $br_inv_file = $ENV{SHAKE_HOME}.'/lib/dam_calc/br_inv.dat';
  $grid_file = $ENV{SHAKE_HOME}.'/data/'.$evid.'/output/grid.xyz';
  $uw_out_file =
$ENV{SHAKE_HOME}.'/data/'.$evid.'/genex/web/shake/'.$evid.'/'.$evid.'_dam_prob.uw';
  $hazus_out_file =
$ENV{SHAKE_HOME}.'/data/'.$evid.'/genex/web/shake/'.$evid.'/'.$evid.'_dam_prob.hazus';
 }
elsif ($method eq 'SA') {
  $br_inv_file = $ARGV[0];
  $grid_file = $ARGV[1];
  $uw_out_file = './dam_prob.uw';
  $hazus_out_file = './dam_prob.hazus';
}
else {
  die "\nThis error should never be seen.  Please report this to the author.";
}
#***********************

#******************
#**  Open the input files and output files.  (open the output files to test if they can be opened/written to)
#******************

open(INV_HDL, "<", $br_inv_file) or die "\nError opening inventory file.", $br_inv_file;
open(GRID_HDL, "<", $grid_file) or die "\nError opening grid file.", $grid_file;
open(UW_OUT, "| $sort_command -r > $uw_out_file") or die "\nError opening pipe to sort for the UW
output file.", $uw_out_file;
#open(HAZUS_OUT, "| $sort_command -r > $hazus_out_file") or die "\nError opening pipe to sort for the
HAZUS output file.", $hazus_out_file;

#******************

#******************
#** Read and store the input
#******************

parse_gridfile();     #Parse the gridfile, read in the meta line and store the grid lines in RAM
if ($grids_used < 4) {
    print "Exiting. Too few lines read from $grid_file\n";
    print "Are you sure your grid file contains eight columns of data?\n";
    exit 0;
}
close(GRID_HDL);
parse_inventory();
close(INV_HDL);

#***************************



#*********************TEST CODE*******************************
#foreach my $br_ref (@bridges)  {
#  print "\n",join(":",@{$br_ref});
#}
#print "\n Last array index: (scalar/dollar-pound)",scalar(@bridges),"/",$#bridges,"\n";
#************************************************************

#This is where the actual calulation of the damage probability occurs.

foreach my $br_ref (@bridges)  {
  my $br_psa03 = 0;
  my $br_psa10 = 0;
  my $br_pga = 0;
  my $gr_lat = 0;
  my $gr_lon = 0;
  my $br_span = 0;
  my $br_year = 0;
  my $br_matl = 0;
  my $br_des = 0;
  my $br_len = 0;
  my $br_nbilen = 0;
  my $br_length = 0;
  my $br_maxspan = 0;
  my $br_length = 0;
  my $UWPd = 0;
  my $HAZUSPd = 0;

  $br_year = @{$br_ref}[3];  #These are here mostly to make parts of the code more clear.  See the table at
the top for more information about what all these arrays are.
  $br_span = @{$br_ref}[5];  #note -- span here is the span type, not the length of the bridge.  Not all of
these are used here.  Their scope is local.
  $br_len = @{$br_ref}[9];
  $br_nbilen = @{$br_ref}[10];
  $br_maxspan = @{$br_ref}[11];
  $br_matl = @{$br_ref}[7];
  $br_des = @{$br_ref}[5];

#*****
# This following section is to deal with bad bridge data, or data outside of the ShakeMap region.  It sets
latitude and longitude in the bridge array to 0 if the
# data from the inventory was wierd or if the bridge was outside of the ShakeMap region.  This is later used
to determine if the bridge should be processed.
# If dam_calc.pl is run on an event where the ShakeMap is wholly outside of Washington, *every* bridge
lat. and lon. is set to 0.  No bridges will be processed,
# and output will be generated that says so.
#*****

  if ((@{$br_ref}[1] < $south_bdry) || (@{$br_ref}[1] > $north_bdry) || (@{$br_ref}[2] > $east_bdry) ||
(@{$br_ref}[2] < $west_bdry)) {
    $gr_lat = 0;
    $gr_lon = 0;
    $br_psa03 = 0;
    $br_pga = 0;
    $br_psa10 = 0;
    $UWPd = 0;
    $HAZUSPd = 0;
    @{$br_ref}[1] = 0;
    @{$br_ref}[2] = 0;
    @{$br_ref}[14] = 0;  #Hazus bridge type.
  }
  else {
    ($br_psa03, $br_psa10, $br_pga, $gr_lat, $gr_lon) = calc_accels($br_ref);

    $processed = $processed + 1;
    $UWPd = UW_prob_calc($br_year, $br_span, $br_psa03);

    $HAZUSPd = HAZUS_prob_calc($br_ref, $br_psa03, $br_psa10, $br_pga);     #insert these variables
here
  }
  @{$br_ref}[15] = $br_psa03;                              #add these values to the bridge array:
$br_psa03, $gr_lat, $gr_lon, $UWPd, $HAZUSPd
  @{$br_ref}[16] = $gr_lat;                                #These values aren't used any more. ($gr_lat and
$gr_lon).
  @{$br_ref}[17] = $gr_lon;                                #They are still there though to avoid confusion.
  @{$br_ref}[18] = $br_pga;                                #See the description of the bridge array at the
beginning of the script.
  @{$br_ref}[19] = $br_psa10;
  @{$br_ref}[20] = $UWPd;
  @{$br_ref}[21] = $HAZUSPd;

}

#These 2 lines print the output file headers.  These headers contain the column labels only.
print UW_OUT "\nPd,DOTID,BRName,BRNum,BRLat,BRLon";
#print HAZUS_OUT "\nPd,DOTID,BRName,BRNum,BRLat,BRLon";

#*****
#Inside the 'foreach' loop, the commented 'print' lines with different output format
#were used to compare data against the Eberhard/Ranf Matlab code output.  These shouldn't
#be needed for any other reason.  They are left there, however, in case they need to be used.
#Note that if they are to be used, that the other output lines should be commented instead.
#*****

foreach my $bridge (@bridges) {
  $| = 1;  # unbuffers output
  if ((@{$bridge}[1] == 0) || (@{$bridge}[2] == 0) || (@{$bridge}[3] == 0)) {
    if ($method eq 'SA') {
      printf(UW_OUT "\n%.5f,%.5f,%s,%s,%s,%.5f,%.5f",@{$bridge}[20],@ {$bridge}[21],
@{$bridge}[8], @{$bridge}[6], @{$bridge}[4], @{$bridge}[1], @{$bridge}[2]);
}
  }
  else {
    printf(UW_OUT "\n%.5f,%.5f,%s,%s,%s,%.5f,%.5f",@{$bridge}[20],@ {$bridge}[21],
@{$bridge}[8], @{$bridge}[6], @{$bridge}[4], @{$bridge}[1], @{$bridge}[2]);
$out_count = $out_count + 1;
  }
}

print "\nBridges in: ", $in_count, " Bridges processed: ", $out_count, " Grid points input: ", $grids_in, "
Grid points used: ", $grids_used, " South bdry: ", $south_bdry, " North bdry: ", $north_bdry, " West bdry:
", $west_bdry, " East bdry: ", $east_bdry, "\n";
close(UW_OUT);
close(HAZUS_OUT);


sub calc_accels {                #Calculates a weighted average of the 4 (or less) gridpoints surrounding the
bridge
  my $bridge_ref = shift(@_);
  my @bridge = @{$bridge_ref};
  my $br_lat = @bridge[1];
  my $br_lon = @bridge[2];
  my @boundpts = (0,0,0,0);
  my @w_ave_pts = ();
  my $wav_denom = 0;
  my $wa_psa03 = 0;
  my $wa_psa10 = 0;
  my $wa_pga = 0;
  my $numpts = 0;
  my $dist = -1;

@boundpts = find_grids($bridge_ref);

  foreach my $grid (@boundpts) {
    if ($grid != -1) {                                                    #if $grid == -1, it's because not all 4
boundry points were needed. ignore.
      $dist = sqrt(($br_lat - @{@shake_pts[$grid]}[1])**2 + ($br_lon - @{@shake_pts[$grid]}[2])**2);
      if ($dist == 0) {                                                   #the grid was right on -- just return it.
          return(@{@shake_pts[$grid]}[5], @{@shake_pts[$grid]}[1], @{@shake_pts[$grid]}[2]);
      }
      else {
          push(@w_ave_pts, $grid);                #push the array index on to this array for later calculation
of the weighted average
          $numpts = $numpts + 1;
      }
    }
  }
  foreach my $point (@w_ave_pts) {
    $dist = sqrt(($br_lat - @{@shake_pts[$point]}[1])**2 + ($br_lon - @{@shake_pts[$point]}[2])**2);
#calculate the denominator for the weighted average
    $wav_denom = $wav_denom + ($dist**$INTERP_POWER);
  }
  foreach my $point (@w_ave_pts) {
    $dist = sqrt(($br_lat - @{@shake_pts[$point]}[1])**2 + ($br_lon - @{@shake_pts[$point]}[2])**2);
    $wa_psa03 = $wa_psa03 + ((($dist**$INTERP_POWER)/$wav_denom) * @{@shake_pts[$point]}[5]);
#acumulate a weighted average for psa03
    $wa_pga = $wa_pga + ((($dist**$INTERP_POWER)/$wav_denom) * @{@shake_pts[$point]}[2]);
    $wa_psa10 = $wa_psa10 + ((($dist**$INTERP_POWER)/$wav_denom) * @{@shake_pts[$point]}[6]);
  }
  return($wa_psa03, $wa_psa10, $wa_pga, @{@shake_pts[@boundpts[1]]}[1],
@{@shake_pts[@boundpts[1]]}[0]);  #return the weighted average psa03 and the last boundry point
(legacy)
}



sub find_grids {
my $bridge = shift(@_);
my $nw_pt = -1;
my $sw_pt = -1;
my $ne_pt = -1;
my $se_pt = -1;
my $beg = 0;
my $end = (scalar(@shake_pts) - 1);
($beg, $end) = lat_search($beg, $end, $bridge);
if (@{@shake_pts[$beg]}[1] == @{@shake_pts[$end]}[1]) {                  #the latitude of the bridge
exactly matched a grid and so there are only 2 other points.
  ($sw_pt, $se_pt) = lon_search($beg, $end, $bridge);
}
else {
  ($sw_pt, $se_pt) = lon_search($beg, int($beg + (($end - $beg)/2)), $bridge); #divide the ones selected
from lat_search() in half -- relies on same num of grids/row
  ($nw_pt, $ne_pt) = lon_search(int($beg + (($end - $beg)/2)), $end, $bridge);
  }
return($sw_pt, $nw_pt, $se_pt, $ne_pt);
}


sub lat_search {
(my $beg, my $end, my $bridge) = @_;
```

```
my $center = 0;
my $grid_max = $#shake_pts;

#*********** TEST CODE -- dumps the status and important variables of this function each call.  For
debugging this function and the search algorithm.
#my $beg_lat = @{@shake_pts[$beg]}[1];
#my $end_lat = @{@shake_pts[$end]}[1];
#$| = 1;        #unbuffers the output for testing purposes
#print "\n latsearch Beg: ", $beg, " ", $beg_lat, " Latsearch end: ", $end, " ", $end_lat;
#print "\n bridge lat/lon: ", @{$bridge}[1], " ", @{$bridge}[2];
#print "\n grid_max : ",$grid_max;
#********** END TEST CODE

my $new_beg = $beg;
my $new_end = $end;

if (($end - $beg) <= 1) {
  my $ref = $beg;
  if (@{$bridge}[1] < @{@shake_pts[$ref]}[1]) {                          #We've picked a point with a lat
slightly over that of the bridge
    while(@{@shake_pts[$new_end]}[1] == @{@shake_pts[$ref]}[1]) {            #These 2 whiles put the
end point at the last gridline of the latitude just below the bridge
      if ($new_end == $grid_max) {last;}
      $new_end = $new_end + 1;
    }
    if ($new_end < $grid_max) {

#***********  TEST CODE -- For testing this function and algorithm in a different place in the loop.
#$beg_lat = @{@shake_pts[$new_beg]}[1];
#my $extra_lat = @{@shake_pts[($new_end + 1)]}[1];
#$end_lat = @{@shake_pts[$new_end]}[1];
#$grid_max = $#shake_pts;
#$| = 1;
#print "\n midloop latsearch new_Beg: ", $beg, " ", $beg_lat, " Latsearch new_end: ", $end, " ", $end_lat, "
extra: ", $extra_lat;
#print "\n midloop bridge lat/lon: ", @{$bridge}[1], " ", @{$bridge}[2];
#print "\n midloop grid_max : ",$grid_max;
#**********  END TEST CODE

      while((@{@shake_pts[$new_end + 1]}[1]) == (@{@shake_pts[$new_end]}[1])) {
            $new_end = $new_end + 1;
            if ($new_end == $grid_max) {last;}
      }
    }
    while(@{@shake_pts[$new_beg]}[1] == @{@shake_pts[$ref]}[1]) {            #This while puts the
beginning point at the first gridline of the latitude just above the bridge
      $new_beg = $new_beg - 1;
      if ($new_beg == 0) {last;}
    }
    return($new_beg, $new_end);
  }
  if (@{$bridge}[1] > @{@shake_pts[$ref]}[1]) {                          #We've picked a point with a lat
slightly under that of the bridge
    while(@{@shake_pts[$new_beg]}[1] == @{@shake_pts[$ref]}[1]) {            #These 2 whiles put the
end point at the first gridline of the latitude just over the bridge
      $new_beg = $new_beg - 1;
      if ($new_beg == 0) {last;}
    }
    if ($new_beg > 0) {
      while(@{@shake_pts[($new_beg - 1)]}[1] == @{@shake_pts[$new_beg]}[1]) {
              $new_beg = $new_beg - 1;
              if ($new_beg == 0) {last;}
      }
    }
    while(@{@shake_pts[$new_end]}[1] == @{@shake_pts[$ref]}[1]) {            #This while puts the end
point at the last gridline of the latitude just above the bridge
      $new_end = $new_end + 1;
      if ($new_end == $grid_max) {last;}
    }
    return($new_beg, $new_end);
  }
  if (@{$bridge}[1] == @{@shake_pts[$ref]}[1]) {                             #We've picked a grid point that has a
latitude equal to that of the bridge
    if ($new_beg > 0) {
      while(@{@shake_pts[($new_beg - 1)]}[1] == @{@shake_pts[$new_beg]}[1]) {
              $new_beg = $new_beg - 1;
              if ($new_beg == 0) {last;}
      }
    }
    if ($new_end < $grid_max) {
      while(@{@shake_pts[($new_end + 1)]}[1] == @{@shake_pts[$new_end]}[1]) {
              $new_end = $new_end + 1;
              if ($new_end == $grid_max) {last;}
      }
    }
    return($new_beg, $new_end);
  }
}
                                                                           #if we haven't narrowed the proximity down to two or less
grid points yet, recurse.

$center = int($beg + ($end - $beg)/2);

#********* NOTE
#Instead of enforcing rounding, lets just abort the recursion in the case of what would otherwise be an
infinite loop consuming all RAM
#This policy makes sure that the grid set keeps converging with successive calls.  Rounding could cause
convergence to cease otherwise.
#*********

if (@{$bridge}[1] > @{@shake_pts[$center]}[1]) {
  ($new_beg, $new_end) = lat_search($beg, $center, $bridge);
  return($new_beg, $new_end);
}
elsif (@{$bridge}[1] < @{@shake_pts[$center]}[1]) {
  ($new_beg, $new_end) = lat_search($center, $end, $bridge);
  return($new_beg, $new_end);
}
elsif (@{$bridge}[1] == @{@shake_pts[$center]}[1]) {
  ($new_beg, $new_end) = lat_search($center, $center, $bridge);
  return($new_beg, $new_end);
}
```

```
}


sub lon_search {
(my $beg, my $end, my $bridge) = @_;
my $center = 0;
my $grid_max = $#shake_pts;

#**********   Test Code -- for debugging the grid finding functions. Outputs grid indices every call.
#$| = 1;
#print "\n lonsearch Beg: ", $beg, " ",@{@shake_pts[$beg]}[0] , " ", @{@shake_pts[$beg]}[1], "\n";
#print "\n Lonsearch end: ", $end, " ",@{@shake_pts[$end]}[0] , " ", @{@shake_pts[$end]}[1], "\n";
#print "\n bridge lat/lon: ", @{$bridge}[1], " ", @{$bridge}[2];
#**********   End test code

if (($end - $beg) <= 1) {
  if (($beg < $grid_max) && ($beg > 0)) {          #Have to make sure we don't overshoot the array index.
(index checking prevents run-time errors.)
    if ((@{$bridge}[2] > @{@shake_pts[$beg]}[0]) && (@{$bridge}[2] < @{@shake_pts[($beg +
1)]}[0])) {    #we've isolated the gridpoint right before that borders the bridge on the lesser longitude side
      return($beg, ($beg + 1));
    }
    elsif ((@{$bridge}[2] < @{@shake_pts[$beg]}[0]) && (@{$bridge}[2] > @{@shake_pts[($beg -
1)]}[0])) {   #we've got the one right after the bridge
      return(($beg - 1), $beg);
    }
    elsif (@{$bridge}[2] == @{@shake_pts[$beg]}[0]) {
      return($beg, $beg);
    }
    elsif (@{$bridge}[2] == @{@shake_pts[$end]}[0]) {
      return($end, $end);
    }
    else {
      $! = 1;
      die "\nError: This should have never been reached in lon_search. Grid index: ", $beg, "\n";
    }
  }
  else {         #Our beginning index is at one of the extremes.
    if ($beg == $grid_max) {
      if (@{$bridge}[2] > @{@shake_pts[$beg]}[0]) {    #we've isolated the gridpoint right before that
borders the bridge on the lesser longitude side
            die "\nError: Trying to overshoot the grid array in Lon_search.";
      }
      elsif ((@{$bridge}[2] < @{@shake_pts[$beg]}[0]) && (@{$bridge}[2] > @{@shake_pts[($beg -
1)]}[0])) {   #we've got the one right after the bridge
            return(($beg - 1), $beg);
      }
    }
    if ($beg == 1) {
      if (@{$bridge}[2] < @{@shake_pts[$beg]}[0]) {    #we've isolated the gridpoint right after that
borders the bridge on the greater longitude side
            die "\nError: Trying to undershoot the grid array in Lon_search.";
      }
      elsif ((@{$bridge}[2] > @{@shake_pts[$beg]}[0]) && (@{$bridge}[2] < @{@shake_pts[($beg +
1)]}[0])) {   #we've got the one right before the bridge
            return($beg, ($beg + 1));
      }
    }
    elsif (@{$bridge}[2] == @{@shake_pts[$beg]}[0]) {
      return($beg, $beg);
    }
    elsif (@{$bridge}[2] == @{@shake_pts[$end]}[0]) {
      return($end, $end);
    }
    else {
      $! = 1;
      die "\nError: This should have never been reached in lon_search. Grid index: ", $beg, "\n";
    }

  }

}
$center = int($beg + ($end - $beg)/2);

#********* NOTE
#Instead of enforcing rounding, lets just abort the recursion in the case of what would otherwise be an
infinite loop consuming all RAM
#This policy makes sure that the grid set keeps converging with successive calls.  Rounding could cause
convergence to cease otherwise.
#*********

if (@{$bridge}[2] < @{@shake_pts[$center]}[0]) {
  return(lon_search($beg, $center, $bridge));
}
elsif (@{$bridge}[2] > @{@shake_pts[$center]}[0]) {
  return(lon_search($center, $end, $bridge));
}
elsif (@{$bridge}[2] == @{@shake_pts[$center]}[0]) {
  return(lon_search($center, $center, $bridge));
}

}

sub usage {
  print "\nUsage : <(inventory file) (gridfile)> or <-event (eventID)>\n";
}

sub parse_gridfile {
  #
  #set the newline character to 0x0a for this file -- it's what ShakeMap produces for some strange reason.
  #then do things just like for the inventory file for the moment -- same sort of array of references to arrays
business.
  #keep in mind that the first row of the 2D array contains the meta info for the grid file.  It probably has
only one column.
  #
   my $tmp_sep = $/;
   $/ = qq{\x0a};                    #grid files have 0x0a as the line terminator.
   @map_params = split(" ", <GRID_HDL>);   #Splits the grid file metaline up into values.
   $west_bdry = $map_params[9];            #sets the boundry variables from the meta line. (These are global
variables.)
   $south_bdry = $map_params[10];
   $east_bdry = $map_params[11];
   $north_bdry = $map_params[12];
   while(<GRID_HDL>) {                       #reads in the grid file, throws out all the lines that don't have 8
objects.
       $grids_in = $grids_in + 1;
       my @gridline = split(" ", $_);
     #    print "\n Just read: ", join(":",@gridline);
       if (scalar(@gridline) == 8) {
                 $grids_used = $grids_used + 1;
                 push(@shake_pts, \@gridline);
       }
       $/ = $tmp_sep;
   }
}

sub parse_inventory {
  #
  #  Read in the bridge inventory.
  #    We'll use an array of references to the arrays that actually hold the data.
  #    Keep an index so that we can assign a number to each bridge for ease in correlation.  This index starts
at 0.
  #  Documentation update: This index isn't really used in this program, but it is there should someone want
to use it.
  #  The index is essentially just a bridge number that is assigned when the file is read in.  They are
sequential.
  #

  my $index = 0;

  while(<INV_HDL>) {
    $in_count = $in_count + 1;
    # Read a line and chop it up by colons. Insert the index in front.

    my @bridge = ($index,split(":"));

    #TEST CODE -- dumps the index and number of elements of the bridge reference array, there should be
a consistent pattern here.
    #print "\n",$index, " ",scalar(@bridge);
    #END TEST CODE

    #
    #push the reference to the array onto the array of references
    #but only if the bridge line has the right number of elements.
    #

    if (scalar(@bridge) == 12) {
      push(@bridges, \@bridge);
      $index = $index + 1;
    }                            #If the data is incomplete, keep a record in the bridge array, but do not process it, set
values to 0.
    else {
      @bridge = ($index,0,0,0,0,0,0,0,0,0,0,0,0);
      push(@bridges, \@bridge);
    }

  }
}
```

```
sub UW_prob_calc {
(my $br_year, my $br_span, my $br_psa03) = @_;
my $Pd = 0;
   if ($br_year <= @uw_years[0]) {
     if (($br_span >= 15) && ($br_span <= 17)) {
            $Pd = Math::CDF::pnorm(log($br_psa03/$lam4)/$xi4);   #pnorm assumes a mean of 0 and a
sigma of 1.  This function isn't documented
            goto(PROBCALCDONE);
     }                                           #in the standard perl docs.  It's only documented within the module
comments.
     elsif (($br_span >= 9) && ($br_span <= 10)) {
            $Pd = Math::CDF::pnorm(log($br_psa03/$lam5)/$xi5);
            goto(PROBCALCDONE);
     }
     else {
            $Pd = Math::CDF::pnorm(log($br_psa03/$lam1)/$xi1);
            goto(PROBCALCDONE);
     }
   }
   elsif ($br_year <= @uw_years[1]) {
     if (($br_span >= 15) && ($br_span <= 17)) {
            $Pd = Math::CDF::pnorm(log($br_psa03/$lam4)/$xi4);   #pnorm assumes a mean of 0 and a
sigma of 1.  This function isn't documented
            goto(PROBCALCDONE);
     }                                           #in the standard perl docs.  It's only documented within the module
comments.
     elsif (($br_span >= 9) && ($br_span <= 10)) {
            $Pd = Math::CDF::pnorm(log($br_psa03/$lam5)/$xi5);
            goto(PROBCALCDONE);
     }
     else {
            $Pd = Math::CDF::pnorm(log($br_psa03/$lam2)/$xi2);
            goto(PROBCALCDONE);
     }
   }
   else {
     if (($br_span >= 15) && ($br_span <= 17)) {
            $Pd = Math::CDF::pnorm(log($br_psa03/$lam4)/$xi4);   #pnorm assumes a mean of 0 and a
sigma of 1.  This function isn't documented
            goto(PROBCALCDONE);
     }                                           #in the standard perl docs.  It's only documented within the module
comments.
     else {
            $Pd = Math::CDF::pnorm(log($br_psa03/$lam3)/$xi3);
            goto(PROBCALCDONE);
     }
   }
  PROBCALCDONE:
   return($Pd);        #returns the UW damage probability.
}
```

```
sub HAZUS_prob_calc {                                                          #this does classification and
damage calculation probability
  my $br_length = 0;
  my $HPd = 0;
  my $mean = 0;
  my $br_htype = 0;
  (my $br_ref, my $br_psa03, my $br_psa10, my $br_pga) = @_;     #insert these variables here
  my $br_year = @{$br_ref}[3];  #These are here mostly to make parts of the code more clear.  See the
table at the top for more information about what all these arrays are.
  my $br_span = @{$br_ref}[5];  #note -- span here is the span type, not the length of the bridge.  Not all of
these are used here.  Their scope is local.
  my $br_len = @{$br_ref}[9];
  my $br_nbilen = @{$br_ref}[10];
  my $br_maxspan = @{$br_ref}[11];
  my $br_matl = @{$br_ref}[7];
  my $br_des = @{$br_ref}[5];
  if ($br_nbilen > 0) {
      $br_length = $br_nbilen;                                                 #select length from the two, set
$br_length
    }
    else {
      $br_length = $br_len;
    }
  $br_htype = HAZUS_classify($br_ref, $br_length);           #Performs the HAZUS classification
  @{$br_ref}[14] = $br_htype;                                #Records the HAZUS type in the array.

  #
  #***** This is the part where it actually does the probability calculation.  I've elected not to make an
array or otherwise reorganize the mean values
  # because some of them are actually calculated in this part itself and are therefore not really constant in
the global context.
  #

  if ($br_htype == 1) {
    $mean = 0.4;
    $HPd = Math::CDF::pnorm(log(($br_psa10/100)/$mean)/$HAZUS_SD);
  }
  elsif ($br_htype == 2) {
    $mean = 0.6;
    $HPd = Math::CDF::pnorm(log(($br_psa10/100)/$mean)/$HAZUS_SD);
  }
  elsif ($br_htype == 5) {
    $mean = 0.25;
    $HPd = Math::CDF::pnorm(log(($br_psa10/100)/$mean)/$HAZUS_SD);
  }
  elsif ($br_htype == 7) {
    $mean = 0.5;
    $HPd = Math::CDF::pnorm(log(($br_psa10/100)/$mean)/$HAZUS_SD);
  }
  elsif ($br_htype == 10) {
    $mean = 2.5*($br_psa10/$br_psa03);
    if (1 < $mean) {
      $mean = 1;
    }
    $mean = $mean * 0.6;
    $HPd = Math::CDF::pnorm(log(($br_psa10/100)/$mean)/$HAZUS_SD);
  }
  elsif ($br_htype == 11) {
    $mean = 2.5*($br_psa10/$br_psa03);
    if (1 < $mean) {
      $mean = 1;
    }
    $mean = $mean * 0.9;
    $HPd = Math::CDF::pnorm(log(($br_psa10/100)/$mean)/$HAZUS_SD);
  }
  elsif ($br_htype == 12) {                    #yes, I realize that this is the same as $h_type == 5. I kept
it this way to make things more clear.
    $mean = 0.25;
    $HPd = Math::CDF::pnorm(log(($br_psa10/100)/$mean)/$HAZUS_SD);
  }
  elsif ($br_htype == 14) {
    $mean = 0.5;
    $HPd = Math::CDF::pnorm(log(($br_psa10/100)/$mean)/$HAZUS_SD);
  }
  elsif ($br_htype == 15) {
    $mean = 2.5*($br_psa10/$br_psa03);
    if (1 < $mean) {
      $mean = 1;
    }
    $mean = $mean * 0.75;
    $HPd = Math::CDF::pnorm(log(($br_psa10/100)/$mean)/$HAZUS_SD);
  }
  elsif ($br_htype == 16) {
    $mean = 2.5*($br_psa10/$br_psa03);
    if (1 < $mean) {
      $mean = 1;
    }
    $mean = $mean * 0.9;
    $HPd = Math::CDF::pnorm(log(($br_psa10/100)/$mean)/$HAZUS_SD);
  }
  elsif ($br_htype == 17) {
    $mean = 0.25;
    $HPd = Math::CDF::pnorm(log(($br_psa10/100)/$mean)/$HAZUS_SD);
  }
  elsif ($br_htype == 19) {
    $mean = 0.5;
    $HPd = Math::CDF::pnorm(log(($br_psa10/100)/$mean)/$HAZUS_SD);
  }
  elsif ($br_htype == 22) {
    $mean = 2.5*($br_psa10/$br_psa03);
    if (1 < $mean) {
      $mean = 1;
    }
    $mean = $mean * 0.6;
    $HPd = Math::CDF::pnorm(log(($br_psa10/100)/$mean)/$HAZUS_SD);
  }
  elsif ($br_htype == 23) {
    $mean = 2.5*($br_psa10/$br_psa03);
    if (1 < $mean) {
      $mean = 1;
    }
    $mean = $mean * 0.9;
    $HPd = Math::CDF::pnorm(log(($br_psa10/100)/$mean)/$HAZUS_SD);
  }
  elsif ($br_htype == 24) {
    $mean = 0.25;
    $HPd = Math::CDF::pnorm(log(($br_psa10/100)/$mean)/$HAZUS_SD);
  }
  elsif ($br_htype == 26) {
    $mean = 0.75;
    $HPd = Math::CDF::pnorm(log(($br_psa10/100)/$mean)/$HAZUS_SD);
  }
  elsif ($br_htype == 28) {
    $mean = 0.8;
    $HPd = Math::CDF::pnorm(log(($br_psa10/100)/$mean)/$HAZUS_SD);
  }
return($HPd);
}

#******
#  End of HAZUS_prob_calc
#******


sub HAZUS_classify {
  (my $br_ref, my $br_length) = @_;
  my $br_year = @{$br_ref}[3];
  my $br_matl = @{$br_ref}[7];
  my $br_des = @{$br_ref}[5];
  my $br_maxspan = @{$br_ref}[11];
  if ($br_maxspan >= 492.13) {
    if ($br_year <= $HAZUS_YEAR) {
      return(1);
    }
    else {
      return(2);
    }
  }
  elsif ($br_matl == 1) {
    if (($br_des >= 1) && ($br_des <= 6)) {
      if ($br_year <= $HAZUS_YEAR) {
	return(5);
      }
      else {
	return(7);
      }
    }
    else {
      return(28);
    }
  }
  elsif ($br_matl == 2) {
    if (($br_des >= 1) && ($br_des <=6)) {
      if ($br_year <= $HAZUS_YEAR) {
	return(10);
      }
      else {
            return(11);
    }
  }
  else {
    return(28);
  }
}
elsif ($br_matl == 3) {
  if (($br_des >= 1) && ($br_des <= 6)) {
    if ($br_year <= $HAZUS_YEAR) {
            if ($br_length <= 65.62) {
              return(24);
            }
            else {
              return(12);
            }
    }
    else {
            return(14);
    }
  }
  else {
    return(28);
  }
}
elsif ($br_matl == 4) {
  if (($br_des >= 2) && ($br_des <= 10)) {
    if ($br_year <= $HAZUS_YEAR) {
            if ($br_length <= 65.62) {
              return(26);
            }
            else {
              return(15);
            }
    }
    else {
            return(16);
    }
  }
  else {
    return(28);
  }
}
elsif ($br_matl == 5) {
  if (($br_des >= 1) && ($br_des <= 6)) {
    if ($br_year <= $HAZUS_YEAR) {
            return(17);
    }
    else {
            return(19);
    }
  }
  else {
    return(28);
  }
}
```

```
  elsif ($br_matl == 6) {
    if (($br_des >= 1) && ($br_des <= 7)) {
      if ($br_year <= $HAZUS_YEAR) {
            return(22);
      }
      else {
            return(23);
      }
    }
    else {
      return(28);
    }
  }
  else {
    return(28);
  }
}
```