NO.73
(1988)

BUILDING RESEARCH ASSOCIATION OF NEW ZEALAND

REPRINT

CI/SfB
(99.78) (Agn)
UDC 694.4.072.9:681.3.06

The development of an Expert System for wall bracing design

W.B.Mugridge and J.G.Hosking

This report originated in work commissioned by BRANZ as part of its ongoing research into knowledge-based systems. The views represented are not necessarily those of the Association

Reprinted from ***Proceedings of the Third New Zealand Conference on Expert Systems,*** Wellington, 11-13 May,1988.

BUILDING RESEARCH ASSOCIATION OF NEW ZEALAND

The Development of an Expert System for Wall Bracing Design

W.B. Mugridge and J.G. Hosking
Department of Computer Science
University of Auckland

Abstract

An expert system is under development which will aid a building designer or building inspector to check that the wall bracing requirements of a code of practice for timber frame houses have been met. It is also planned that the system will guide the user in designing the wall bracing to meet those requirements.

A prototyping approach has been taken in the development of the system. The first stage was to complete a code checking system, followed by a system to guide the user through the process of providing bracing to meet the load requirements. The checking system evolved from a simple system which ignored many of the complexities of the code. Difficult issues, such as dealing with complex house shapes, were left until the simple cases were handled. *Class Language* classes proved invaluable here in structuring the knowledge base. The modularity meant that it was usually straightforward to revise the knowledge base structure as further complexities were handled.

Several aims were established for the project, and conclusions are already available for most of these.

1. Introduction

An expert system, *WallBrace*, is under development as part of our continuing collaboration with the Building Research Association of New Zealand (BRANZ). This collaboration is due to our joint interest in the general application of expert systems to codes of practice and to design issues which are of interest to the building industry (Whitney, 1987; Dechapunya, 1987; Hosking et al, 1987a,b; Fowkes et al, 1988; Hosking et al, 1988).

Designers and building inspectors have difficulties in interpreting the wall bracing requirements of *NZS 3604: 1984 Code of Practice for Light Timber Frame Buildings*. Many Local Authorities use this code and require a wall bracing schedule with a building permit. It was decided by BRANZ that an expert system would save time for designers, building inspectors, and BRANZ advisory staff who are often asked to help with the wall bracing requirements. Such a system would also aid in the design of wall bracing, a *routine design* problem (Brown and Chandrasekaran, 1986).

Routine design can occur when the experience of a designer permits design choices to be made without the need to iterate in finding a solution to a design problem (Brown and Chandrasekaran, 1986). Without such experience, the search for a design solution can be difficult or impossible (Mittal et al, 1986; Mostow, 1985). Hence much work in the area of design automation is concerned with codifying experience so as to constrain a design problem to be routine (Maher and Fenves, 1985). Little other work has centred on the use of expert systems for codes of practice (Fenves and Garrett, 1986; Rosenman and Gero, 1985).

A manual example of this design process is the development by BRANZ of the NZS 3604 Wall Bracing Calculation Sheets, which spell out the design procedure for wallbracing design in such a fashion as to make the application of Clause 6.3 of NZS 3604 a routine design problem. These calculation sheets are extensively used by building designers, yet some still experience difficulty in making use of the information.

This paper begins with a discussion of the written code and the initiation of a project to build an expert system. The development of the first stage is mostly complete and is discussed in some detail. Brief introductions are given for the following two stages. Finally, the aims of the project are assessed, and various conclusions drawn from this work.

2. The Code and the Research Project

2.1 The Code

The wall bracing requirements are specified in *NZS 3604: 1984 Code of practice for light timber frame buildings not requiring specific design*. These requirements provide a simplified version of the loadings code, *NZS 4203: 1984 Code of practice for general structural design and design loadings for buildings*. The majority of buildings constructed in New Zealand are of light timber frame construction.

The code specifies how to use some basic information about a building, such as the height and type of roof, to calculate the loading on the building in two orthogonal directions. The loadings are calculated for both earthquake and wind loadings. The earthquake loading is based roughly on the storey height, the surface area and weight of the roof (categorised as light or heavy), and the earthquake zone of the building (as given in Table 1 of the NZS 3604, shown in Fig. 1). The wind load is based on the wind exposure zone, the exposed roof and wall surfaces, and takes account of the number of storeys, the storey height, and the slope and height of the roof. The calculations involved represent a rough mapping from some attributes of a building to a measure of loading which can be used to ensure that adequate bracing has been provided to resist horizontal loads in the building.

All loadings are expressed in *bracing units* (B.U.'s). A 2.4 m long section of 2.4 m high wall which is supported by a bracing element (consisting of a diagonal brace with gib-board on one side) supports 100 B.U. (or 5 kN) applied horizontally if it does not distort in that direction by more than 8 mm. The bracing element here is said to supply 100 B.U.

Once the loadings for the building (expressed in B.U. units) are known, appropriate bracing elements are allocated to external walls and internal bracing lines. These must at least total the number of B.U.'s required due to the loadings. In addition, there are bracing requirements which are applied locally to a wall. For example, an external wall must have at least 10 B.U. per metre of wall with additional bracing requirements under some special conditions.

In order to keep the code simple, it appears that a number of assumptions, which are not spelled out in the code, are made about a building. The major assumption is that the building is uniform in structure, so that all parts of it have the same number of storeys, with the same height, and the same type roof structure, height, and slope. For example, Table 11B (as shown in Fig. 1) makes the assumption that all the storeys are of the same height.

These uniformity assumptions are not always met, so they must be dealt with in some way. The difficulty of dealing with non-uniform buildings was not clearly realised until after some progress on the project. This has led to the project being much larger than originally anticipated.

2.2 The Problem

Some people have difficulties in understanding and applying the code. (The following examples of questions were supplied by the BRANZ expert.) Problems arise from users not knowing how to interpret the tables or clauses of the code: "I don't have a wall at 5m from the exterior wall - what can I do?". The aim of the code is to spell out the bracing requirements. It provides no guidelines on how to meet those requirements. For example, it gives no assistance with questions such as "How can I get enough bracing to meet the total required?" and "Can a ceiling diaphragm be used to fix this?".

The immediate aim, therefore, was to provide a system which could aid a user in the allocation of wall bracing to a building. On completion of the task, the system was to produce a summary report for the loadings on the building and the bracing that had been provided; this could then be attached to a plan of the building for submission to the local authority along with the building permit application.

The resulting system was also to play an educative role, making explicit the steps and calculations that were being carried out. This would enable a user to gain a better understanding of the wall bracing requirements and of the steps involved in meeting them.

2.3 The Research Aims

Several aims were established for the research project beyond the immediate aim of constructing a usable expert system. Some of these aims reflect our joint interest with BRANZ in the application of expert systems to codes of practice and to design.

Wall bracing was considered likely to be a good initial problem to examine techniques for the construction of knowledge-based systems for design. This was because some of the expertise has already been made explicit within the written code and because the application appeared to be an example of straightforward routine design.

The project would also provide further experience in building expert systems based on codes of practice. Some tentative conclusions about the suitability of a code of practice as the basis for an expert system were made in Hosking et al (1987). It was considered that the complex conditional nature of the DZ4226 (the draft fire code used) was a principal factor in making that code suitable for such an approach. A secondary consideration with DZ4226 was the extensive commentary provided with the code. It is appropriate that the wall bracing project be used to determine whether those conclusions should be revised.

Another aim of this work was to discover the benefits of using an expert systems approach over using a conventional programming language, such as Fortran. It appeared that the wall bracing problem is sufficiently procedural that a solution in a conventional programming language was feasible. However, it was considered that a knowledge-based approach would offer a number of benefits over a conventional approach, both in the construction of the system and in the facilities provided to the user, such as the ability to try alternative approaches. A related aim was to identify attributes of a problem which would enable a choice to be made between a conventional approach and an expert systems approach.

A final aim was to foster the further development of *Class Language* (Mugridge et al, 1987; Hamer et al, 1988) by providing experience in a new application area - that of routine design.

2.4 The Research Project

WallBrace was commissioned by BRANZ in 1987. The intended users were building designers and inspectors, who would be familiar with the terminology of the code.

There were to be two major phases in the construction of *WallBrace*. The first phase was to create a system to aid a building designer or building inspector to check that the wall bracing requirements of a code of practice for timber frame houses have been met. The second phase was to extend the system so that it guides the user in designing the wall bracing to meet those requirements.

3. Development of Stage One: a Simple Checker

The aim of the first stage was to develop a checking system for the wall bracing requirements of NZS 3604. It was thought that once the checker was complete, it would be easier to develop the design aspects of the project. It would also enable the checking system to be evaluated while the design aspects were being added.

As the project developed, it became clear that the written code did not spell out the loadings and bracing requirements for buildings of complicated shape. The code essentially deals with simple buildings with a minimal variety of building height and roof structure. For example, it does not spell out how to deal with a U-shaped building in which the three main parts of the building have different numbers of storeys and different types of roof.

It was decided that the original Stage One should be split into two parts, with the new Stage One being concerned only with simple buildings which could be handled directly with the code as written. It became clear that considerable knowledge acquisition would be required in order to handle the complex buildings to be dealt with in the new Stage Two.

Stage One is complete to the final prototype stage and is in the process of refinement for delivery: developing the user interface, and improving the wording of questions, displayed information, and explanation.

The prototyping approach taken in Stage One has been used throughout the project. In the early phases a subset of the task was chosen for initial implementation. The advantages of this were:

- early feedback was available to the expert on the form of the expert system
- the expert could point out inappropriate dialogue and incorrect logic
- the knowledge engineer could gradually become familiar with the complexities of the code.

Using a prototyping approach can be enhanced by making use of a programming language which provides features for modularity. *Class Language*, designed to provide effective modularity, was used in this project. Evaluation of this aspect of *Class Language* would provide valuable feedback to further its design .

Stage One went through a number of phases, including: preliminary investigation, initial design, encoding tables, using a goal-directed approach, and addressing user-interface issues.

3.1 Preliminary Investigation

The first step, carried out in a preliminary investigation, was to understand what was required from an expert system for wall bracing. This understanding was gained by working through a number of examples, determining the wall bracing requirements for particular buildings. This helped in understanding the instructions with the BRANZ wall bracing sheets and in clarifying such issues as the relationship between the wind and earthquake loading on the width and length of a building.

The first meeting with Joe Ten Broeke, the BRANZ expert, clarified the many questions that were raised. The code does not specify the reasoning behind various provisions; it is written assuming an appropriate background on the part of the reader. It also tends to spell out what is and is not allowed, without always clarifying how to meet those requirements. It was apparent that there would be problems with the interpretation of Table 11 (as shown in Fig.1) in the code in particular.

In order to begin the process of prototyping early, a dummy dialogue of a user with the future wall bracing system was constructed and shown to the expert. This dialogue was valuable in isolating inappropriate assumptions that we had made about various aspects of the task and the code. In addition to the dialogue, a list of assumptions and questions had been prepared. The consultation with the expert led to feedback on issues of:

- the scope of the system
- the terminology to be used
- the form of the dialogue
- clarification of many points and questions about the written code.

Questions were raised about how to handle complicated building shapes. It was decided at an early stage to deal initially with simple shapes and to consider complex cases later. This would allow us to gain a better understanding of the code and the process of applying it, and it would allow the expert to gain a clearer idea of what an expert system could provide.

In our experience, it is good to start with simple examples when beginning to develop an expert system. A written code, for example, has to handle a wide variety of cases and so is difficult to digest as a whole. Looking at simple examples means that requirements that are unusual can be ignored until the commonly-applicable aspects are understood.

3.2 Initial Design

With a basic understanding of the intent of the written code, the next step was to create an initial design of the knowledge base. This was begun by isolating the various significant objects (such as building, roof, and wall) that are mentioned. It has been our experience that much of the initial work in building a regulation-based expert system is centred on object representation.

Many objects and their properties are mentioned in the written code, so it was necessary at this stage to determine which were central to the system and which were peripheral or could be ignored in the early stages. For example, the size, weight and slope of the roof play a significant role in determining the wind and earthquake loadings on a building (see Fig. 1). On the other hand, some components play a peripheral role in bracing, and so could be left out of the initial design.

Sometimes obvious objects may not be modelled in a straightforward manner. For example, the roof was eventually modelled as three objects, representing:

- general information about the roof (such as the height, type, and size);
- information relevant to wind loading ACROSS the roof (such as slope); and

• information relevant to wind loading ALONG the roof.

Another approach, in the initial stages of design, would be to first list all the objects, and their properties, that are mentioned in the code. This is an approach advocated by Fenves et al (1987) in applying various computer-based techniques to the process of regulation formulation. This has the advantage that all objects are made explicit early in the project, and the disadvantage that many of the objects will not be relevant to the expert system development.

The structure of the knowledge base at this stage is shown in Fig. 2. This shows the component structure of the various parts of a building, represented as *classes* in *Class Language*.

3.3 Encoding Tables

It would seem likely that it would be relatively simple to encode the information from a table in a code of practice. Unfortunately, this is often not the case. The sorts of problems that arise are:

• The purpose of the table is unclear. This occurs, for example, in the draft fire code DZ4226 (Hosking et al, 1987b).

• Information is missing. This occurs, for example, in the draft loadings code, DZ4203 (Hosking et al, 1988).

• There are ambiguities in the terms used. This occurs in all the regulation-based systems we have constructed.

• It's not clear whether it is possible to interpolate or extrapolate the figures given. This occurs in Table 11 of NZS 3604, in the wall bracing provisions.

• A set of examples is given and the user is expected to choose the example most similar to their case. Often it is not spelled out that the user is expected to interpret the table in this way. This occurs, for example, in the draft fire code DZ4226 (Hosking et al, 1987b).

• Important information is "hidden" away in footnotes to tables. For example, in Table 11 of NZS 3604 (as shown in Fig. 1) it is left to a footnote of the table to specify that: "The maximum number of bracing units is to be the sum of those for wind acting on the walls and wind acting on the roof."

Such problems can arise through:

• General cases not being considered by the code writers;

• A conflict between the need to be simple for the simple cases, and the need to have a complete specification for all cases;

• Assumptions being made about what is usual. For example, the assumption that all the storeys in 3-storey house will be the same height, as in Table 11 of NZS 3604.

3.4 A Goal-Directed Approach

Once the overall class structure had been mapped out, it was appropriate to turn to the active, problem-solving aspects of the system. A goal-oriented approach was then taken by considering the final report to be produced and the initial information displayed to the user. From this it was possible to work back to the information required from the user. The structure of the report produced was based on the subcomponents of *building*, which generate the different parts of the report (see Fig. 2 for the class structure). For example, the *building* class adds information about the building and its environment to the report, while the *roof* class adds information to the report about the roof.

As the production of the report is a sequential process, the report was encoded as procedures in the various classes. Additional *properties* were added to the classes as they were required by the report. This meant defining the questions to be asked of the user, or the rules to be used in finding a value for a property. For example, class *storey* was extended with the following properties: *height*, *length*, *width*, *buildingPlanArea*, and *seismicLoad*. This clearly separates those parts of the system which are concerned with the **logic** of the problem-solving from those parts which are concerned with the overall **sequence** in which it must be carried out. It is appropriate to encode these in different forms: the first as rules which are used when needed, and the second as procedures which are called. In Class Language, the procedures make use of property values in producing the summary and displaying information to the user; this has the affect of initiating backward-chaining on the rules when a value is unknown.

This approach has also been taken in the *Seismic* project (Hosking et al, 1988). We have found that working backwards from the final results of the system focusses attention on the **logic** of the problem-solving, rather than on the **sequence** in which it happens to be carried out.

3.5 User Interface Issues

A number of issues related to the user-interface were addressed. The dialogue must be sensible to the user, both in the language used and in the order in which the consultation occurs.

Improving the Prompts
Once buildings were being considered that had several storeys, it was necessary to provide better information in the prompts so that the actual storey and the direction (along or across the building) were spelled out. This is especially so when there are several wings, with several storeys in each wing. For example, in the following, the second question is better than the first:

> What is the height of this storey (in metres) ?
>
> What is the height of the top storey of the library wing (in metres) ?

Providing information during the consultation
It was found to be useful to provide feedback to the user about the overall intention behind groups of questions and when intermediate results have been determined. Providing general information would enable the system to play an educative role with a designer who was not familiar with the code. For example, the system determines the average height of the roof and whether the earthquake or wind loading is greater in a particular direction. This process and the results should be made explicit to the user.

Focussing the Dialogue
Once the basic logic of the system was encoded, it was found that the dialogue was sometimes confusing. It was necessary to alter the sequence of questions that resulted from

the logic provided. This was done through the use of *procedures* and *when procedures* in *Class Language* (Hamer et al, 1988).

Shortcuts
In some cases, it is possible to avoid asking many questions when it is inappropriate. For example, asking the general question "Are diaphragms used in the building?" can avoid repeated questions about diaphragms. Similarly, the system asks whether all storeys have the same dimensions; this avoids asking for the dimensions for every storey. Providing shortcuts usually means asking the general question at the appropriate level and passing the answer as a parameter to the specific cases.

Explanation
Explanation text was added late in the development of the system.

3.6 Modularity of the Knowledge Base

Modularity of the knowledge base is critical to the success of a prototyping approach. It must be possible to continue reorganising the knowledge base as the knowledge engineer's understanding of the problem evolves. If there is minimal or inappropriate modularity, it is difficult to restructure the system to keep the design clean and clear. In the development of *WallBrace* there were several massive reorganisations of the knowledge base which resulted in no changes at all to most of the classes in the system. At other times, classes had to be split into several or recombined to reflect the changing design.

Modularity in *Class Language* is provided through the following mechanisms:

- separate classes and associated rules and procedures for separate objects
- parameterised classes
- classification
- inheritance

Using Classes
Class Language is an object-oriented language, so that modularity is achieved through modelling the different classes of objects found in an application. The classes are defined according to the useful aspects that have to be modelled. Some objects will correspond to obvious objects in the world, such as *roof*, while others will be rather abstract, such as *loadDirection*.

Rules and procedures are associated with classes, so the information that is tightly coupled is found together. Often changes to a class will not have any effects outside that class.

Parameterised Classes
The calculations for the wind loading ALONG and ACROSS the building follow the same logic. It was inappropriate to duplicate this logic for the two directions, so a class was defined which generalises the aspects of both directions, having suitable parameters to distinguish the two directions for the user. Classes allow suitable abstractions, reducing the overall detail and making explicit the commonality between different parts by creating a more general class.

Classification
The next step was to encode the wall bracing elements as classes. The designer has a choice of six types of bracing elements, as listed in Table 20 of NZS 3604, each with different properties and restrictions, as specified in clause 6.9. This was clearly a case for the use of classification rules.

Inheritance
In considering the types of bracing element, it is clear that there are a number of common

attributes. These included: the amount of *bracingProvided* by the element, the *bracingType*, the *length* of the bracing element, the *minimumLength* allowed, and the *bracingPerMetre* provided by the chosen bracing element. These were all declared as properties in the most general class, *bracingElement*, inherited by the classes representing each type of bracing element, and were given values by specific rules in those specialisation classes.

Being object-oriented, *Class Language* provides for powerful modularity based purely on objects. However, it is unable to provide modularity according to other programming paradigms such as the procedural one. In addition, there are different types of knowledge that are encoded in a common form in the system. For example, rules in a single class are used to encode both clauses of the code and techniques for reducing the number of questions asked of the user. During development, the expert could be interested in the encoding of the code clauses, but is less likely to be interested in the other aspects which are like conventional programming. A general mechanism of viewing the knowledge base in different ways would be most useful.

4. Development of Stage Two: Checker of Complex Buildings

As mentioned above, the wall bracing requirements of NZ3604 make some implicit assumptions about the uniformity of the building being braced. This has meant that *WallBrace* has had to be extended beyond the code in order to handle non-uniform buildings.

Fig. 3 shows an example of a non-uniform building. It was not initially clear how to handle such buildings, other than that the provisions of the code should be able to be applied to the individual sections of the building. However, the code itself does not specify how to combine the loading information to calculate the overall loading for the building. A means of load combination had not been formalised at all, so it was necessary in Stage Two to create such a formalisation.

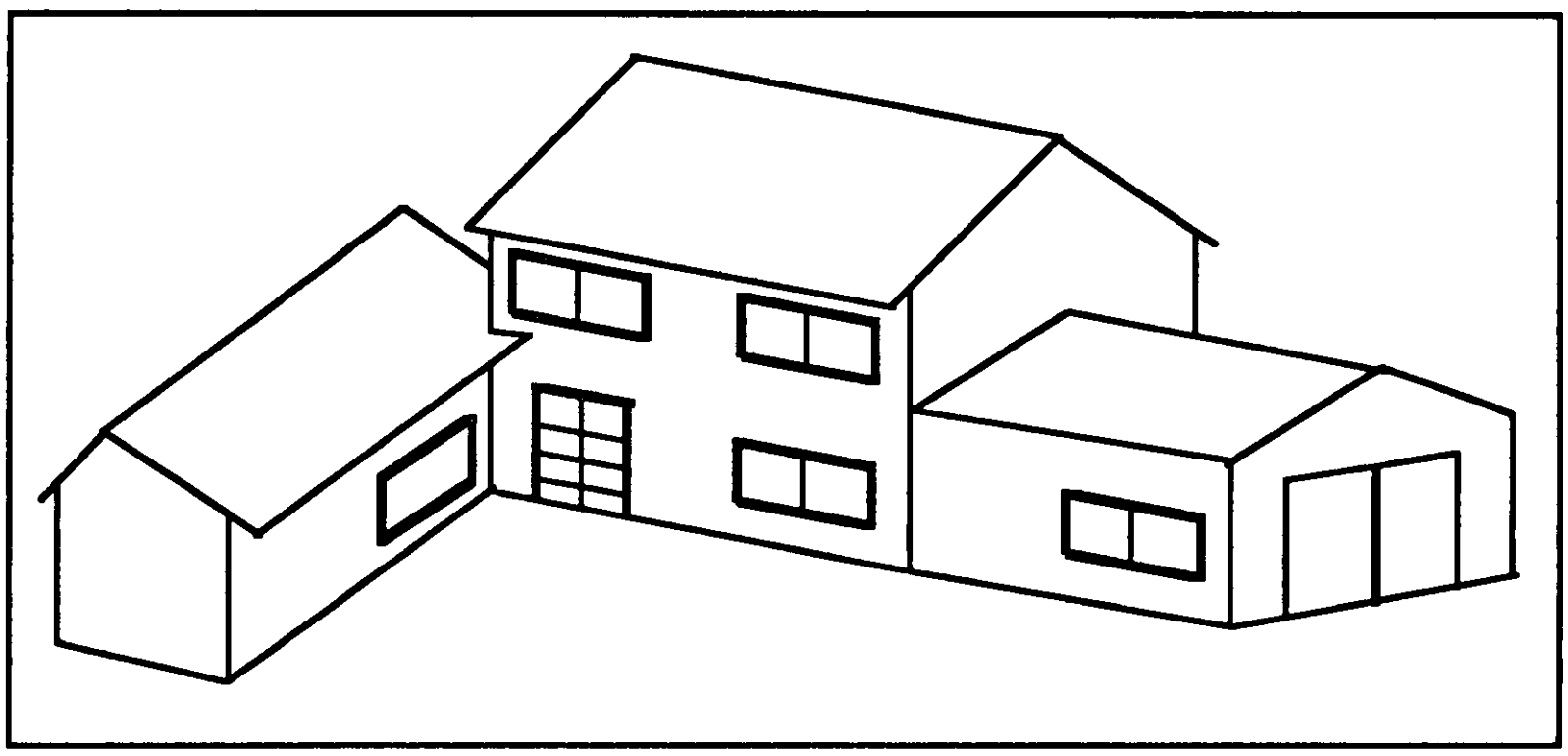

Figure 3. Non-uniform Building.

The expert was able to specify how to combine loads for particular buildings, based on general spatial/connectivity considerations, but had no general method of dealing with the wide range of possible building shapes. Examples of buildings were prepared and supplied to the expert for discussions, in order to formulate a general strategy for load combinations. Fig. 4 shows one of the sample sheets of examples created for the expert. The general approach in creating the example buildings was to take a simple case and perturb it in various ways into cases which could not be handled. A progression of perturbations meant that the cross-over point between using one approach and another could be found.

Early approaches centred on categorising the general shape of the building, but this proved to be quite unsatisfactory. After much discussion, we were able to formulate a general approach which the expert thought was correct. This method required much more detailed information about a building than previously, in order to calculate the combined loadings. However, in most cases buildings will be rather simple in structure and so the amount of information is not great. Where the building is complex, the user is likely to be in greater need of assistance and so will be prepared to enter the information required.

The development of Stage Two to a prototype is mostly complete, although buildings with a variety of storey heights are not yet handled. The spatial reasoning involved has been straightforward to encode once the method was worked out. Further effort will be required to ensure that the method used is general enough. Alternatively, those buildings which cannot be handled by the system will have to be explicitly disallowed.

5. Stage Three: Design Aid

Stage Three will provide assistance to the user in various aspects of wall bracing:

(a) Laying out bracing lines within the building, within the constraints of the code. This will require further spatial reasoning in the system.

(b) Distributing bracing through the building. This will depend on how much the building parts are tied together, so general heuristics will have to be developed to decide how much the loading of one part can be distributed to connected parts of the building. Bracing distribution has to be a multi-step process because meeting all the local bracing requirements may not mean that the global bracing requirements of the written code are met.

(c) Allocating bracing elements to individual walls in the building. This will be based on such factors as the load, the wall space available, and the type of wall.

The development of (a) is under way, while knowledge acquisition is in the early stages for (b) and (c).

6. Assessing the Project Aims

A number of aims were enumerated for the project: gaining experience with design-based systems and the further application of expert systems to codes of practice; deciding when to use an expert systems approach rather than a conventional programming approach; and facilitating the further development of *Class Language*.

It is too early to comment on our experience with building a design-based system because the third stage of the project is still in progress. However, this project has provided us with further useful experience in building expert systems and in particular those based on codes of practice.

With regard to codes of practice, the project confirms some results from earlier work on *FireCode* (Hosking et al 1987). In several respects, the development of *WallBrace* has been similar to the development of *FireCode*, in that the written code appeared straightforward but that there were difficulties with the encoding of tables used in the two codes of practice involved. However, we must revise our earlier assessments concerning the suitability of a code as a basis for an expert design system. DZ4226, as used in the *FireCode* project (Hosking et al, 1987), seemed suitable because of the complex conditional nature of the code. This type of complexity is not the case in the *WallBrace* and *Seismic* projects. In these cases, the suitability of the code for an expert system approach is based on the need to provide additional support to the code, which has to be provided by an expert. The written

code provides an initial formalisation but is incomplete. In this respect DZ4226 has been formalised to a much greater extent, with the result that the code is bigger and apparently more difficult to use because more is spelled out. This greater completeness means there is less need for an expert in the construction of an expert system.

The advantages of a knowledge-based approach over the written code are: the ability to provide explanation, the ability to focus on relevant subsections of the code, and the ability to try alternatives while retaining consistency.

Along with the concurrent project on *Seismic*, this project has led to further ideas about knowledge acquisition. Our experiences of knowledge acquisition are covered elsewhere (Hosking et al, 1988).

One of the aims of the project was to isolate means of choosing between a conventional approach and a KBS-approach. It has become very clear that the expert systems approach being taken in building *WallBrace* is more appropriate than a conventional programming approach. Although the wall bracing requirements of the written code appeared initially to be reasonably straightforward, once work was under way it became clear that the representational issues were far from trivial and thus better handled with a language with good representational facilities.

Prerau (1985) addresses this issue from a bottom-up point of view: the selection of an appropriate application domain for using an expert systems approach. However, many of his points also apply to the decision as to whether to use a conventional programming approach (and language) or to use an expert systems approach (and language). Many of the points that Prerau makes are concerned with other aspects of selecting an application, such as the suitability of the expert, the amount of managerial support, and whether an expert systems approach is likely to be feasible.

We consider some selected points from Prerau (1985) which are relevant to the choice between the two approaches and assess them with respect to *WallBrace*:

(1) "The domain is characterized by the use of expert knowledge, judgement, and experience". In this case, it was clear that an expert was often needed because of the many requests that BRANZ receive for seminars about wall bracing and for specific help with the code. A partial formalisation of bracing exists in the written code, but it is inadequate for the task. We have found that some written codes do not offer adequate help with suitable procedures for applying the code. For example, *Seismic* is based on a code which assumes considerable expertise in the process of applying the code.

(2) "The task primarily requires symbolic reasoning". Some numeric computation is required when applying the code, but it is quite simple. Most of the intellectual effort in the use of the code is concerned with making qualitative decisions. This seems to be the usual case with the codes of practice that we have studied.

(3) "The task requires the use of heuristics ... or may require decisions to be based on incomplete or uncertain information". The written code itself essentially prescribes a set of heuristics, such as the various categories that are used (such as the categories of roof weights and roof slopes). It was not obvious at the beginning of the project that additional heuristics were required. Further heuristics were introduced by the expert for handling the loadings on complex buildings, those that were not handled by the written code itself. Incompleteness and uncertainty became an issue with complex buildings, because of the gross assumptions that needed to be made to apply the code. The code attempts to model loadings and bracing in a very general way, while trying to avoid detailed models. This is to make it simple enough to be used in routine design of timber frame buildings. The unfortunate result is that the assumptions that are used to simplify the model are not spelled out - it's difficult to use the code with a complex case.

(4) "The task is neither too easy (taking a human expert less than a few minutes) nor too difficult (requiring more than a few hours for an expert)". The task for a simple building is certainly too easy for an expert, but a novice may still require help because they do not understand how to apply the code. The task for a complex building is by no means trivial, but is not too difficult.

(5) "The amount of knowledge required by the task is large enough to make the knowledge base developed interesting. If it is too small, the task may be more amenable to another approach - e.g. a decision tree". This is likely to be difficult to assess before knowledge acquisition has begun. In the case of wall bracing, it was very likely at the beginning that the task would be large enough, especially with the design component. This has certainly turned out to be the case.

(6) "The task is sufficiently narrow and self-contained..". This may be difficult to assess in general. It appeared at the beginning to be the case for the wall bracing task. It has become clearer during the project that a lot of spatial reasoning is required.

(7) "The number of important concepts (e.g. rules) required is bounded to several hundreds". The number of classes of objects of importance in a code of practice could be counted, to give a guide. Also, the number and complexity of clauses would give an indication of the number of rules. However, our experience shows that an expert system based on a code of practice is most likely to contain considerable knowledge which is not derived from the code but from the expert. This was certainly the case with *Seismic* (Hosking et al, 1988).

A conventional programming language, such as Fortran, provides minimal support for modularity. An unconventional language like Smalltalk (Goldberg and Robson, 1983) would certainly be more suitable than Fortran, but still lacks a number of language features which are critical to building an expert system: the separation of logic and control, the provision of explanation, and allowing the user to explore "what-if" questions.

A final goal of the project was to encourage the development of *Class Language* (Hamer et al, 1988). This project has made comprehensive use of many features of *Class Language*, due to the complexity of representation required. This is especially so in stage two, where objects have to be viewed in different ways (sections under storeys and above them). The project has also provided feedback and ideas for the further development of the language and the system, both in simple ways (such as requiring better formatting facilities) through to more radical proposals (such as redesigning the manner in which explanation and other user-interface issues are handled).

In addition, it is clear that high quality graphics are essential in providing a good user-interface to knowledge-based systems, and especially those that deal with space. Work is underway to provide graphical capabilities in *Class Language*, both for the construction of expert systems and for use in delivery systems.

7. Conclusions & Future Work

7.1 Conclusions

Our approach has been to start to develop the knowledge base before understanding all the details about the project. This is one of the documented advantages of a prototype-oriented approach to building software systems (and expert systems in particular): start building it to get a clearer idea of the whole. It was necessitated particularly because the written code itself was problematical enough without our trying to handle complex shapes. This prototyping approach requires disciplined changes to keep the design well-structured and clear. This means that modularity is extremely important in supporting the continuing refinement of an expert system.

The basic steps, at each of the many phases of prototyping, were as follows. First, interviews with the expert clarified such issues as the appropriate interpretation of tables in the written code. Second, the class structure of the knowledge base was revised to take account of the expanded functions required. This meant drastic changes to the overall structure at several stages. Third, a goal-directed approach was taken, by working from the requirements of the final report backwards through the code requirements to the questions to be asked of the user. Fourth, the user interface was improved, through finding the appropriate level for questions and through the use of shortcuts to reduce the number of questions asked. Fifth, the expert reviewed dialogues of the running system, pointing out problems and suggesting improvements.

The system has become surprisingly complex, given the apparent simplicity of the written code. This complexity arises from: irregularities in the written code (such as in Table 11), providing suitable shortcuts to the user, getting the sequence right for the user, and extending the system beyond the written code.

7.2 Future Work

Codes of Practice: Design
We plan to continue our research into the application of expert systems to codes of practice by constructing an expert system which aids in the design of concrete columns in buildings. The type of design here is somewhat more complex than that required for *WallBrace*, because of the need for several cycles of design-analyse, in which the analysis phase provides feedback to refine the design.

In addition, we plan to extend the capabilities of *Class Language* to make it better suited to design applications. It is expected that the column design application will provide further useful feedback to the continuing language design.

Modularity
Any linear document, such as a book or code of practice or program, has a particular major structure imposed on it which has a significant effect on its order. There are potentially many different organisations of a single document, each suiting different users. Regulations are often organised in ways which make them difficult to use when attempting to apply them to a particular case. One of the benefits of providing regulations in the form of an expert system is that the system can dynamically organise the regulations to suit the task of applying the code to a particular case (Hosking et al, 1987a). Fenves et al (1987) attempt to address this issue of resolving many potential organisations when creating a set of regulations.

While *Class Language* provides effective object-oriented modularity, it would be useful to be able to view the knowledge base in a number of different ways. For example, those parts of the system concerned with code clauses could be viewed independently from a procedural view. Smalltalk (Goldberg and Robson, 1983) provides a browser, which provides a type of viewing system. A mechanism like this is also provided in hypertext systems (Delisle and Schwartz, 1987). Cordy et al (1987) discuss a knowledge-based tool of this form to aid in program development. Unfortunately, it is extremely difficult to allow significant changes through such views, so there still needs to be a single underlying organisation, within which all structural changes are made.

Acknowledgements

The authors gratefully acknowledge the long-term collaboration and financial support of BRANZ, and in particular the expertise provided by Joe Ten Broeke and the support of Dr Haris Dechapunya. We also gratefully acknowledge John Hamer for creating *Class Language*.

References

Brown D C, Chandrasekaran B, 1986, Knowledge and control for a mechanical design expert system, *IEEE Computer*, July 1986.

Cordy J R, Eliot N, Robertson M, 1987, TuringTool: a knowledge-based user interface to aid in the maintenance task, *Report 87-193*, June 1987, Department of Computing Science, Queen's University at Kingston, Kingston, Canada.

Dechapunya A H, 1987, Environment for BRANZ research and development in KBS, *Proc. Second New Zealand Conference on Expert Systems*, Auckland, New Zealand, 1987, pp 59-67.

Delisle N M, Schwartz MD, 1987, Contexts - a partitioning concept for hypertext, *ACM Trans. on Office Information Systems*, **5**, (2), pp 168-186.

Fenves S J, Garrett J H, 1986, Knowledge based standards processing, *International Journal for Artificial Intelligence in Engineering*, **1** (1), pp 3-14.

Fenves S J, Wright R N, Stahl F I, Reed K A, 1987, *Introduction to SASE: Standards Analysis, Synthesis, and Expression*, Report NBSIR 87-3513 U.S. Department of Commerce, National Bureau of Standards, May 1987.

Fowkes A H R, Sharman W R, Dechapunya A H, 1988, The development of marketable knowledge-based systems, *Proc. Third New Zealand Conference on Expert Systems*, Wellington, New Zealand, 1988.

Goldberg A, Robson D, 1983, *Smalltalk-80: The Language and its Implementation*, Addison-Wesley, Reading, Mass, 1983.

Hamer J, Hosking J G, Mugridge W B, 1988, The evolution of Class Language, *Proc. Third New Zealand Conference on Expert Systems*, Wellington, New Zealand, 1988.

Hosking J G, Mugridge W B, Buis M, 1987a, Expert systems for regulations and codes, *Proc. Second New Zealand Conference on Expert Systems*, Auckland, New Zealand, 1987, pp 37-47.

Hosking J G, Mugridge W B, Buis M, 1987b, FireCode: a case study in the application of expert system techniques to a design code, *Environment Planning and Design B*, **14**, pp 267-280.

Hosking J G, Lomas S, Mugridge W B, 1988, The development of an expert system for seismic loading, to be presented at the *Knowledge-Based systems in Civil Engineering Symposium*, Monash University, Melbourne, Australia, August 1988.

Maher M L, Fenves S J, 1986, *HI-RISE: A Knowledge-based Expert System for the Preliminary Structural Design of High Rise Buildings*, Report No. R-85-146, Department of Civil Engineering, Carnegie Institute of Technology, Carnegie-Mellon University, Pittsburgh, Pensylvania.

Mittal S, Dym C L, Morjaria M, 1986, PRIDE: an expert system for the design of paper-handling systems, *IEEE Computer*, July 1986.

Mostow J, 1985, Towards better models of the design process, *Artificial Intelligence Magazine*, Spring 1985, pp 44-56.

Mugridge W B, Hamer J, Hosking J G, 1987, Class Language - a language for building expert systems, *Proc. Second New Zealand Conference on Expert Systems*, Auckland, New Zealand, 1987, pp 173-180.

Mugridge W B, Hosking J G, Buis M, 1988, FireCode: an expert system to aid building design, in Newton P, Taylor M A P, Sharpe R (Eds), *DESKTOP PLANNING: Advanced Microcomputer Applications for Physical and Social Infrastructure Planning*, to be published by Hargreen, Melbourne, 1988.

Prerau D S, 1985, Selection of an appropriate domain for an expert system, *Artificial Intelligence Magazine*, Summer 1985, pp 26-30.

Rosenman M A, Gero J S, 1985, Design codes as expert systems, *Computer Aided Design*, **17**, pp 399-409, 1985.

Whitney R S, Knowledge based systems for building technology: strategic aspects, Proc. Second New Zealand Conference on Expert Systems, Auckland, New Zealand, 1987, pp 5-11.

Copy 1

B17182
0025521
1988
The development of an exp
ert system for wall braci

BUILDING RESEARCH ASSOCIATION OF NEW ZEALAND INC.
HEAD OFFICE AND LIBRARY, MOONSHINE ROAD, JUDGEFORD.

The Building Research Association of New Zealand is an industry-backed, independent research and testing organisation set up to acquire, apply and distribute knowledge about building which will benefit the industry and through it the community at large.

Postal Address BRANZ, Private Bag, Porirua

ISSN 0111–7459