# Managing the operational risks of user-developed software

**Louise Pryor**
www.louisepryor.com

**Abstract**: Operational risk is increasingly coming under the spotlight, and software risk is an important part of it. In this paper I discuss issues relating to user-developed software, which is increasingly prevalent in a number of important organisational areas. The advantages of user-developed software are clear to see, and are primarily based on the use that is made of the expertise of the users who do the developing, while the disadvantages and risks are likewise based on gaps in that expertise. Software risks arise from the way that software is used as well as possible bugs in the software itself, and often merge into issues of competitiveness. Risk identification is a vital component of risk management, and a technique based on one used in chemical process plants for many years is applicable to software processes too. The development and maintenance of user-developed software is one of the important aspects of its use, and is as subject to risk as any other. Many of the risks can best be addressed at the development stage through the application of well thought out processes and standards. Finally, the main significance of bugs is probably not their presence, but the effort that goes into ensuring their absence. Software engineering techniques can assist with this effort, giving the same level of comfort for less work
**Keywords**: Operational risk, user-developed software, risk management, risk identification, software engineering.

## 1 Introduction

Software risk is a component of operational risk (see the working party report presented to this conference) and as such is becoming increasingly prominent in corporate consciousness. However, it is not as glamorous as some other operational risks, and this lack of glamour applies especially to the type of risk that this paper addresses: the risks attaching to user-developed software. These risks tend to be high frequency, low impact risks rather than the low frequency, high impact risks, such as fraud, that make the headlines. Indeed, it is often a matter of opinion whether these risks are better viewed as issues of efficiency and competitiveness.

In the end it may not matter how they are viewed. Both risks and competitiveness issues should be treated in the same way: they should be identified, assessed, and then a decision made as to what action, if any, should be taken. The approach of not trying to distinguish between risk and competitiveness was followed by King (2001), who defines operational risk as "a measure of the link between a firm's business activities and the variation in its business results". King argues that addressing operational risk thus increases value by reducing the variability of earnings.

## 2 User-developed software

The risks of user-developed software have been recognised for a long time. As long ago as 1982 a paper appeared with the title "Caution: User-developed systems may be hazardous to your organization" (Davis 1982). The prevalence of user-developed software is much greater now than it was in the early 80s, so the warning is even more relevant now than it was then. Nowadays spreadsheets are used for internal planning, financial reporting and pricing; modelling systems, programmed by the user through parameters or code snippets, are also widely used; macros and small programs written in VB or other languages are increasingly common.

For example, many insurance companies do their reserving using a combination of

- Data extraction programs provided by their IT departments;
- Specialist reserving or financial modelling products;
- Spreadsheets;
- "Glue" programs consisting of VB macros or other code;
- Small scale database programming.

Typically, all except the first of these are user-developed. The specialist reserving or financial modelling products are driven through parameters or code snippets, put together by the actuarial team, who also build spreadsheets and other programs. The database used is often part of an office applications suite, programmed by a member of the team, rather an industrial strength database programmed by a specialist.

Reserving is not the only function that uses this type of system. Other functions that often use similar combinations of software are pricing and planning. This situation is not specific to insurance companies, either: many financial analysts build their own systems, and complex modelling is performed through user-developed systems in many fields such as transport and health,

The advantages of such systems being developed by their users are based on the expertise of those users. If they build the systems themselves, they can be sure that they perform the calculations correctly, without having to explain sometimes rather abstruse mathematical techniques to someone without their own technical background. The lack of this extra layer between theory and implementation provides time savings to the users in terms of both explaining the requirements to the developer, and in terms of reviewing the finished system to ensure that it does indeed meet the requirements. The systems can be updated more rapidly, and can reflect the actual user requirements in terms of behaviour as well as algorithms.

The disadvantages are also based on the expertise of the users, which is unlikely to include significant exposure to software engineering techniques. This means that they are often aware of potential problems, such as the high probability of bugs, but don't necessarily know the best way of handling them. Sometimes they may not even be aware of possible problems; for example, few user-developers explicitly consider usability issues as they build their systems.

## 3 Software risk

There are three main areas of software risk.

- The software may produce erroneous results.
- The software may fail to produce any results at all, or fail to produce them by the time that they are needed.
- The cost of producing the results (usually measured in person hours) may be greater than necessary.

The latter area of risk is often a result of trying to avoid one of the first two, or of handling the problem caused by one of the first two. For example, if a bug is found in a program the risk of producing erroneous results is lessened, but the cost of producing the results is increased by the time taken to fix the bug.

### 3.1 Examples of risks

To get some idea of what sort of risks might be present with user-developed software, we'll look at an example. Consider a system for calculating reserves. A program supplied and operated by the IT department extracts the basic data from the administration system into text files, one for each line of business. A specialist reserving product reads the text files, performs some processing, and writes out the results to another set of text files. A set of spreadsheets is then used to perform the final analysis and allow the actuary to make the decision

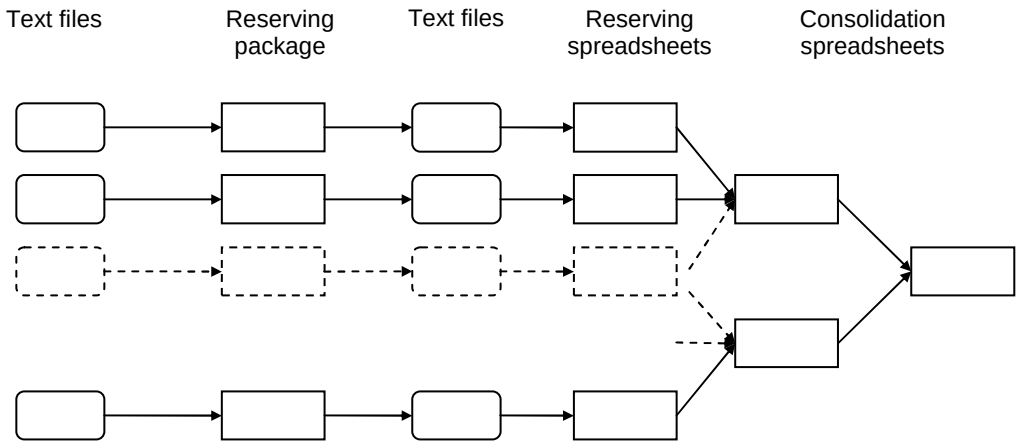



Figure 1. A reserving system

as to what reserves to set for each line of business. Finally another set of spreadsheets, linked to each other and the analysis spreadsheets, is used to consolidate all the results (see Figure 1).

The most obvious thing that can go wrong is that the wrong results are produced. There are numerous ways in which this can happen, just a few of which are:

- Bugs in the spreadsheet formulae;
- Mistakes in the reserving system parameters;
- User error in the reserving spreadsheets;
- Erroneous data flow: one of the initial text files going through the wrong reserving parameters, or one of the intermediate text files being processed by the wrong reserving spreadsheet, or even a spurious file finding its way into the system;
- A consolidation spreadsheet picking up the wrong numbers from one of its predecessors;
- Format errors in the text files;
- Unmet assumptions about the ordering of data in text files or spreadsheets;
- Misspellings in text fields, leading to lookup errors.

The second thing that can go wrong is that no results are produced. Again, there are numerous possible causes, including:

- Power failure;
- Hardware failure, with inadequate backups;
- Invalid or missing data causing a system crash;

- Invalid or missing user inputs causing a system crash;
- Absence of the only person who knows how to run the system.

Also included in this group of problems is the case when results are not produced in time, because the process takes too long. This could be due to computational inefficiency, for example, or because manual procedures take longer than the available time.

The third thing that can go wrong is that the results may cost too much to produce. Cost is generally measured in terms of person hours. Increased costs often arise as problems in the first two groups arise, or are thought possible, and steps are taken to overcome or avoid them. Another cause of increased costs (as well as of erroneous results) is poor usability. If the software has not been developed with the needs of users in mind (other than the user doing the developing, of course) it may be time consuming and confusing to enter the correct inputs.

Most of these risks are not specific to a particular component of the system; they arise out of the operation of the system as a whole. This is especially clear when we note that many of the problems are more likely to occur if changes have been made to part of the system; if a spreadsheet has been revised, for example, other spreadsheets that link to it may no longer pick up the correct data.

It is thus important to consider software risk in relation to the system as a whole, not only in terms of single components of the system.

### 3.2 Risk identification

A good risk identification process is a vital component of an effective risk management strategy. It is important that no possibly source of risk is omitted. This is best done using a combination of systematic consideration of the system being analysed with brainstorming. The identification process should include people that know the system well, as well as outsiders who come to the problem with a fresh set of eyes.

A piece of software is never used in isolation; it is always part of a larger system or process, which may contain other pieces of software as well as manual procedures, and it is the risks of the process as whole that should be identified.

A structured process, called Hazops, has been used and developed over the last forty years or so for identifying potential hazards and operability problems in chemical process plants. It fits the description above, and can be adapted to software processes quite easily.

The Hazops process is based on the notion of specifying the intended behaviour of a process, then looking at all the possible deviations, how they might be caused and whether they have adverse effects. It can (and should) be performed at various levels: on the process as a whole, on individual components and, depending on the circumstances, at the code level.

Hazops is especially useful because, as well as identifying risks, it provides a framework within which to analyse their causes and decide how to control or mitigate them. An added advantage is that it provides a good audit trail of the risk management process.

### 3.3 Risk mitigation and control

After risk identification, the next stage is to analyse the risks and see if they can be controlled or mitigated. This can be done by looking at the potential causes, and seeing if it is possible to prevent the cause or to sever the link between cause and adverse consequence.

Looking at some of the examples from section 3.1 above, we can see that many of the risks could be avoided, or at least made less likely, by writing better software and designing better overall systems: *ie*, by paying more attention to issues of risk during the development stage. For example, bugs in the spreadsheet formulae and mistakes in the reserving system parameters should be detected and fixed early on. The design of the spreadsheets can lessen the chance of user error, and the design of the system as a whole can make erroneous dataflow impossible, or at least less likely. Assumptions can be made explicit, and some simple checks can be included in the code.

### 3.4 System development and maintenance

An important aspect of user-developed software is that it is indeed developed and maintained. In this it is no different from any other software: it is unlikely to remain unchanged over a long (or even short) period. Indeed, this is one of the reasons for using user-developed software to start with: the ease of changing it to meet evolving requirements.

Changes can vary in significance and effort from changing the time period that is analysed by altering a single parameter, through adding new products, to changing the whole basis of calculation.

There are risks inherent in the development and maintenance process that can affect the overall availability of the software, as well as the costs of producing it:

- Any changes made, whether to fix existing problems or to add new functionality, may introduce new problems;
- Changes may take much longer than expected;
- Changes may result in the unavailability of the system, if the changes can't be made to work and previous versions are unavailable.

It is important that the development and maintenance process is treated in the same way as any other use of the software, and the risks inherent in it identified and managed in the same way.

## 4 Development process and standards

We saw in the last section that the development and maintenance of user-developed software plays a major role in managing its risks:

- What is done during development and maintenance can make a big difference to the risks of using the software;
- Development and maintenance can be seen as one of the modes of use, so is subject to risks in the usual way.

It is therefore vital that development and maintenance are carried out in such a way as to reduce the overall software risks. The field of software engineering addresses these issues and more, and there are many software engineering techniques that can usefully be applied to user-developed software.

An important aspect of many techniques is that they emphasise the vital role of process and standards. The idea is that if there are routine ways of performing development tasks, important steps are less likely to be omitted and significant issues less likely to be overlooked.

Standards can cover more or less any aspect of development, from the procedures to follow when starting to change a spreadsheet or model, the tests to perform and the documentation that should be produced, through to particular coding constructs that should be used or avoided.

Standards should be designed to perform specific functions, so that following them will mean that certain problems are avoided, or that the software being developed will have certain (desirable) characteristics. The use of such standards leads to easier review as well as easier development. If the standards are well designed, the reviewer need only check that the standards have been followed, rather than check from first principles that the problem has been avoided or that the desirable characteristic is present. The developer, on the other hand, needn't start from scratch when deciding how to implement an

algorithm or produce a desired behaviour. By limiting their choice as to available techniques to use, the standards make the decision easier.

It is important to use only standards that really do serve a useful purpose; otherwise they will not be followed. A couple of pages that are actually used will do much more good than a whole book full of detail that is rigorously ignored.

### 4.1 Development process: example

For example, a simple procedure for making changes to a model might consist of the following:

- The purpose of the desired change is clearly specified (for example, to fix a problem or to introduce new functionality);
- The person with responsibility for maintaining the model approves the change;
- A copy is taken of the existing (working) state of the model;
- The copy is changed to fix the problem or introduce the functionality;
- The changed version is thoroughly tested, both for the effects of the change and for any unwanted and unforeseen side effects;
- The changes made are reviewed by somebody other than the person who implemented it;
- The changed version replaces the existing version of the model.

Obviously this is somewhat under specified; for example the procedure may differ if more than one change can be made at once.

This process ensures that there is always a working version of the model, and if used properly can help make sure that modifications are made quickly and effectively. Large potential modifications should be split into several smaller ones, each implemented separately. The smaller the actual changes that are made, the easier it is to detect and fix any problems that they introduce.

Testing and review serve different purposes, but are equally important (and both depend on functional specifications). Testing is the only way to determine whether a system of any sort actually does what it is meant to do. However, it can only flag the existence of problems; on its own it cannot point out how they should be fixed. Review, on the other hand, is extremely good for pointing out how any potential problems can be fixed, and can be used to check for adherence to standards as well as for functional behaviour.

### 4.2 Coding standards: examples

Coding standards can be used to lessen the possibility of many different types of problems: errors caused by users inputting invalid values, inefficient computation, and unintended changes among them. They can also be used to facilitate the reviewing and testing process: a spreadsheet designed with the reviewer in mind can take significantly less time to review when changes are made to it, for example.

Standards are generally of two types: techniques that should be avoided, and techniques that should be used.

In spreadsheet programming, techniques that should be avoided might include the use of white text on a white background for intermediate workings, repeated recalculations of the same intermediate results, overly complex formulae and the use of macros except in clearly defined circumstances. Techniques that should be used might include data validation for user inputs, sheet protection to avoid unintended changes, and standard layouts for certain types of calculations.

These examples have been chosen pretty much at random, and would not be particularly useful in the absence of an overall policy for spreadsheet construction, considering such issues as what the desirable characteristics of spreadsheet are, who the users are likely to be, what types of changes are likely to be made in the future, and how much effort should be expended on the spreadsheet.

## 5 Conclusion: the true significance of bugs

People get very concerned about the high frequency of bugs in their software, and think of this as a major component of software risk. This is understandable; it has been said that every piece of software contains bugs. Recent estimates indicate that between 20% and 80% of spreadsheets contain significant errors (Panko 2000). Methodological studies indicate that other forms of financial modelling are likely to show similar error rates. However, the studies that produce these estimates may not reflect the situation on the (actuarial) ground. On the whole, the high level of concern about the possibility of bugs leads to a great deal of effort going into detecting and fixing them. Although some undoubtedly slip through, they are unlikely to result in significant errors, which would be spotted at the level of overall reasonableness.

However, this does not mean that bugs are an insignificant aspect of software risk. On the contrary, in my view first impressions are correct and they are indeed one of the major sources of software risk. Their significance arises

because so much effort toes into detecting and fixing them; they thus consume a very high proportion of the resources that go into the development of user-developed software. If this effort could be reduced, while keeping the same level of comfort as to the absence of significant bugs, productivity could be much greater.

The real problem is that this is an area in which actuaries, and other user developers, lack the expertise that would enable them to reduce this effort. The field of software engineering provides a battery of techniques that can be used during the development process that will make the software that is produced both less likely to contain bugs in the first place, and easier to review and test so that bugs can be detected more easily. It is a truism that the earlier a bug is found, the less time it takes to fix. Therefore, the use of techniques that lead to the earlier detection of bugs can give significant improvements in productivity, as can the use of techniques that lead to the presence of fewer bugs to start with.

The use of software engineering techniques can help in other ways too, such as making the software that is produced more usable by others, and more computationally efficient. To use an analogy, it is possible to make yourself understood in a foreign language if you don't know the grammar as long as you have a good vocabulary. It may be difficult to get your sense across, but with enough effort and patience it is usually possible. This is exactly the situation in which many user developers find themselves: they have a good vocabulary, in that they have a good knowledge of the programming language they are using (spreadsheet formulae, model parameters), so that they can build working systems. However, if they also knew the grammar provided by software engineering techniques they could build the systems more efficiently, and the systems would be more effective.

**References**

Davis, G. B. (1982). *Caution: User-developed systems can be hazardous to your organization*. Proceedings of the Seventeenth Annual Hawaii International Conference on System Sciences, Honolulu, Hawaii.

King, J. L. (2001). *Operational risk: Measurement and modelling.* Wiley.

Panko, R. R. (2000) *What we know about spreadsheet errors.* Working Paper, Honolulu, HI 96822: Information Systems Department, College of Business Administration, University of Hawaii.