# The Integration of a Formal Safety Analysis into the Software Engineering Process: An Example from the Pacemaker Industry

D. Santel, C. Trautmann, W. Liu

Instruments Design Group
Medtronic, Inc.
Minneapolis, MN, USA 55432

**Abstract**

A continuing increase in the application of software based systems to safety critical processes has necessitated the integration of safety into the software development process. An example from the cardiac pacemaker industry of this safety oriented software design methodology is provided. Safety consideration begins upon product description with the hazard analysis whereby potential hazards are classified according to their criticality and qualitative probability of occurrence. The completed hazard analysis determines the software safety requirements which specify what the system *will not* do. A system hazard cross check matrix is defined that serves to direct the design team to those areas requiring more scrutiny. Finally, traceability matrices provide a verifiable link from safety requirements through design to the actual code. This methodology should serve to minimize hazards in the final software product.

**Introduction**

Recent years have seen an increase in the application of software based systems to safety critical processes. The results of a software fault in such an application could be loss of property, injury or death. It becomes imperative to offer a system that presents an acceptable level of risk while preserving the benefits it was designed to provide. In the biomedical device industry, particularly with life sustaining systems like the cardiac pacemaker, the potential morbidity or even mortality that software faults can cause is apparent. In this paper, a rationale and method for integrating safety into the software development process is presented.

**Software Safety -- A Historical Perspective**

Identifying software safety as a discrete design goal is a relatively new idea in the cardiac pacing industry where hardware designs were favored over software designs until recently. Other industries, however, have been employing these methods for several years. The literature has examples of software safety implementations from the nuclear power industry[1], the aerospace industry[2][3], and the transportation industry[4]. Software systems are controlling potentially hazardous processes in these industries because their benefits outweigh the potential risks and usually there are no alternate solutions[5].

**Software in the Pacemaker Industry**

Software based cardiac pacing systems are becoming more prevalent. The pacing system consists of a pacemaker, a pacing lead and a pacemaker programmer. The pacemaker, which generates the cardiac stimuli, and the pacing lead or wire that delivers the stimuli to the heart are permanently implanted in the patient. The programmer provides a means to communicate with the pacemaker after implant so that its status may be relayed to the physician and so that parameter values (such as pacing heart rate) in the pacemaker may be modified in response to the patient's physiologic needs.

Some of today's pacemakers contain microprocessors and are software based. Many others are still based on analog and digital circuitry. All modern day programmers, however, are based on microprocessor designs. This paper shall focus on the software safety issues concerning the pacemaker programmer.

A diagram of a pacemaker programmer is shown in Figure 1. The programmer can be thought of as a software based communication device. User input is solicited through a keypad and user output is directed to a liquid crystal display (LCD) and/or to a printer. Pacemaker inputs and outputs are conducted via a radio frequency (RF) generating and receiving head that rests on the patient's chest over the implanted pacemaker. This RF head can downlink new pacing parameter values to the pacemaker and can receive uplinked pacing parameter values from the pacemaker. These values can be displayed on the LCD or directed to the printer.

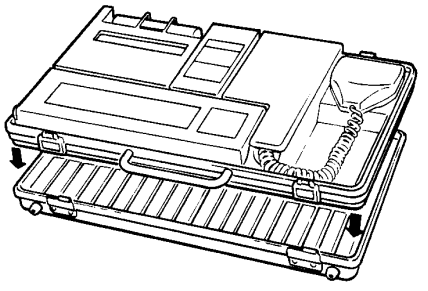

Figure 1 -- Pacemaker Programmer

Many factors dictate that current day pacemaker programmers be microprocessor based. Fifteen years ago pacemakers were "programmed" via transdermal needles inserted into a pacing rate adjusting potentiometer located on the pacemaker. Just several years ago pacemakers could not uplink data to the programmers. Communication with pacemakers was essentially one-way. Pacemakers of this era had few adjustable parameters besides heart rate, stimulating pulse amplitude and width. Most pacemaker manufacturers had few pacemaker models available. Pacemakers of today can have over one hundred adjustable parameters. Tables of legal parameters and allowable values for each must be stored in the programmer's memory. Current pacemakers are able to store long term data detailing how the pacemaker and lead are performing. In order to preserve the battery life of the pacemaker, minimal processing occurs in the pacemaker itself. Rather, this data is uplinked to the programmer where algorithms analyze and display it. A microprocessor is necessary to provide the computational and control power required for these tasks.

**Software Development Changes**

Safety has always been an important, albeit not a focal, part of the software development process in the pacemaker industry. To the physician, safety is paramount in a life sustaining device like the pacemaker. Historically, all engineers and designers assumed safety as a part of their responsibilities. Safety issues were always discussed when product features or functionality were considered. Still, a standard software development methodology consisting of a requirements phase, a functional and detail design phase, a coding phase and a verification phase has been employed, without a separate identifiable safety effort. Several changing factors in recent

years have obviated this view of software safety, forcing a focused and specific safety effort throughout the development process.

As pacemaker programmers became more reliant on software, the type of engineer needed to develop them has changed. Where once a product could be designed with electrical and mechanical engineers , there are now additionally software engineers and computer scientists. While the strengths, size and diversity of the project team have increased, the increasing complexity of the design effort makes it more difficult to insure safety is considered adequately.

An adjunct to the increased degree of specialization among design team members, is the decreased interdisciplinary experience most engineers are able to bring to the development effort. Fewer engineers today have both the technical depth in their particular area of engineering and a thorough understanding of the pathophysiology involved in cardiac pacing. The net result is a technically specialized engineer who does not possess the breadth of background to adequately assess what constitutes a safety hazard in a programmer or pacing system.

**Impetus for Change**

In addition to the obvious changes in the technology of pacemaker systems, other factors are forcing a change in the way the pacemaker industry views software safety. The Food and Drug Administration (FDA), which reviews and approves medical devices before they can be distributed to the market, has developed a great deal of software acumen in recent years. Just last year the FDA published a technical report for its investigators on software development activities. In this report, the FDA has recognized that it is becoming, "...extremely important for the Agency to verify that proper controls were employed to assure the correct performance of the computer system prior to its implementation..."[6]. While the FDA claims that this document is not intended to serve as a vehicle to establish software development methodologies, the message is clear that the FDA is serious about uncovering software defects that could justify regulatory action[7].

**The Integration of Safety Into the Software Development Process**

It has been suggested that safety must be treated as a specific issue in the software development process with responsibility for it assigned and not assumed[5]. The first step of the software safety analysis should occur upon completion of the marketing product description. This step employs the hazard analysis, borrowed from safety analyses of hardware systems[5] [8]. This analysis classifies potential hazards according to their criticality and probability of occurrence. There is an inherent difficulty in quantifying this probability of occurrence of a hazard in software. While the large amount of historical information on the reliability of hardware components allows probability to be quantified with some degree of accuracy, the usually custom configuration of software precludes rigorous probabilistic quantification[9]. Qualitative probability statements are used instead, with typical categories being, "frequent", "occasional", "reasonably remote", and "extremely remote"[5]. In the example of the pacemaker programmer, software faults could lead to the programmer erroneously sending data to the pacemaker that would prevent the pacemaker from stimulating the heart adequately to initiate a heart beat.

The completed hazard analysis dictates a list of software safety requirements. The hazard analysis details the unsafe states of the system which should be translated to software safety requirements. Unlike functional requirements, safety requirements specify what the system *will not* do as well as what it will do.

Safety is considered at each ensuing phase of software development. Once the functional and subsystem design is complete, the software safety analysis can begin. Houston[8] proposed constructing a system hazard cross check where interactions between system hazards and functions are displayed in a matrix (Figure 2). This system cross check matrix has the most reliable functions and least severe hazards near the origin and the least reliable functions and most severe hazards farthest from the origin. Marking the intersection of a hazard row and a function column indicates a system hazard interaction. Intersections that occur far from the matrix origin require a thorough examination of the corresponding function and hazard, while intersections near the origin would merit less attention.

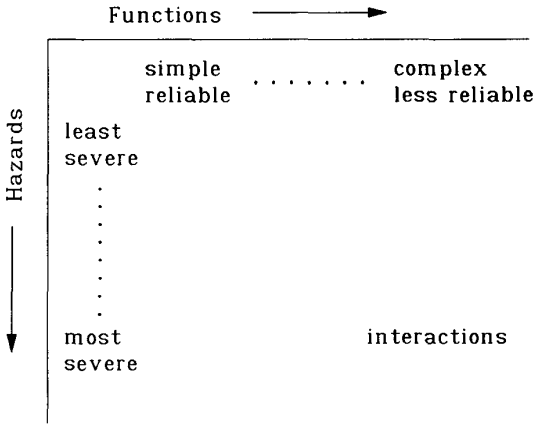


Figure 2 -- System Cross Check Matrix

To insure that safety hazards are considered throughout the design process, we have developed a set of traceability matrices that shows the interrelationship between all software safety and non-safety requirements, the functional design, detail design, coding and system verification. These matrices allow the tracking of all software requirements through the development process (figure 3). Requirements are traced through to each subsequent level of the design. System verification is also performed at all levels of the design. These matrices will allow forward and reverse traceability from requirements through verification.

| |
|---|
| Requirements to Functional Design |
| Requirements to Detail Design |
| Requirements to Code |
| Requirements to System Verification |
| System Verification to Functional Design |
| System Verification to Detail Design |
| System Verification to Code |

figure 3 -- Traceability Matrices

**Where are we now?**

We currently have completed a hazards analysis for a pacemaker programmer. Obviously, the success of the entire safety analysis is contingent upon the careful and thorough consideration of the safety hazards. The determination of the hazards and their criticality was made by engineers in house that had significant clinical exposure to cardiac pacing. In cases of doubt, clinical experts were consulted.

Traceability matrices are currently being developed using a database software package for several software based products. The entire software design team as well as independent reviewers from other internal projects will be reviewing the matrices for completeness at the end of each stage of development.

**Summary**

Software safety analysis is an accepted part of the software design methodology in the nuclear and aerospace industries and is gaining acceptance in the biomedical instrumentation industry. The importance of software safety in life sustaining devices like the cardiac pacemaker dictates that safety analysis be included in these software product development efforts.

Employing hazard analyses, specifying safety requirements and tracing all requirements through the entire development process helps to minimize the number of safety hazards that exist in the final product. While these techniques can not be expected to eliminate every hazard and error, they offer the developer and the end user increased confidence in the integrity of the software based product. As the premise of the cardiac pacing industry is to sustain and enhance life, it seems especially appropriate that we do all that is practical to insure that our devices help and never hinder the patients who depend on them.

## References


1. Gmeiner, L., and Voges, U., "Software Diversity in Reactor Protection Systems: An Experiment," *Safety of Computer Control Systems*, R. Lauber, ed., Pergamon Press, Elmsford, N.Y., 1980, pp. 75-80.

2. Eccles, E.S., "Software for Flight Critical Digital Engine Controls", *ASME*, 80-GT-119, 1980.

3. Helps, K.A., "Some verification tools and methods for airborne safety-critical software", *Software Engineering Journal*, Vol 1, No. 6, 1986, pp. 248-53.

4. Cribbens, A.H., "Microprocessors in railway signalling: the solid-state interlocking.", *Microprocessors & Microsystems*, Vol 11, No. 5, 1987, pp. 264-72.

5. Leveson, N.G., "Software safety: Why, what and how", *Computing Surveys*, Vol. 18, No. 2, 1986, pp. 125-63.

6. "Software Development Activities -- Reference Materials and Training Aids for Investigators", US Department of Health and Human Services, Public Health Service, Food and Drug Administration, July, 1987.

7. "The GMP Letter", Washington Business Information, Inc., No. 91, August, 1987, pp. 2-7.

8. Houston, M. Frank, "What do the simple folk do? Software Safety in the Cottage Industry", *Supplement to Proceedings of Compass '87 Computer Assurance Conference*, IEEE Press, 1987, pp. s-20 - s-24.

9. Gloe, G., "Inspection of process computers for nuclear power plants", in *Proceedings of IFAC SAFECOMP '79*, Pergamon, Elmsford, N.Y., 1979, pp. 213-18.