

Safety and Reuse

Bill Frakes
Computer Science Department
Virginia Tech
frakes@cs.vt.edu

Some definitions of safety from Nancy Leveson.

“Safety is freedom from accidents and losses”

RISK is the hazard level combined with (1) the likelihood of the hazard leading to an accident (sometimes called danger) and (2) hazard exposure or duration (sometimes called latency).

“A hazard is a state or set of conditions of a system (or an object) that, together with other conditions in the environment of the system (or object) will lead inevitably to an accident (loss event).” Leveson p. 177 A hazard is defined with respect to the environment of the system or components. What constitutes a hazard depends upon where the boundaries of the system are drawn.

Now reuse.

Software reuse is the use of existing software knowledge or artifacts to build new systems. (see, e.g., Frakes and Kang, 2005)

The problems with reuse in terms of safety apparently arise from inserting an asset into a new system environment without sufficient understanding of the context required for its use, as was the case in the Therac accident.

This raises several questions –

1. How might context be represented?
2. Is it possible to assure that the contextual specification is correct and complete?

But what exactly is context? In the reuse community one of the best-known uses of the term is in the 3Cs model of reuse design. In the 3Cs model context means the environment necessary to reuse a component. A common example is the stack data type. In the three Cs model, an implementation neutral (if there is such a thing) formal specification of the stack will be its concept, a particular implementation such as using arrays in C might be its content, and the environment needed to run the component, say the gnu C compiler running on Free BSD Linux might be its environment.

Context can be represented using a natural language, as in the stack example, or a formal language, or some combination of the two. As is well known, the problem is that natural languages are ambiguous and formal languages are hard to understand and may lack

expressiveness. Recent work on software dependability is based on the assumption that purely formal specifications of a system are impossible [Knight, 2006]. Specifying environmental context also a long term, and unsolved, problem is software testing.

It is unclear that this is the problem for all types of reuse. In generative reuse, for example, the main problem may be a fault in the generator itself. [Smidts, 2002] This problem was identified in work on reliability, and the exact role of reliability in the reuse and safety problem is unclear.

The following table summarizes some factors in software disasters, and will serve as a basis of discussion.

	Therac	Ariane	Patriot	Shuttle	Toilet
Embedded					
Size					
Reuse Type	Carryover Component	Carryover Component			
Language		Ada			
Error	Task coord.	Error Handler	FP error	Variable initialization	UI
Reuse?	Y	Y	N	N	N

References

N. Leveson, Safeware: System Safety and Computers, Reading, MA, Addison-Wesley, 1995.

Frakes, William and Carol Terry. "Software Reuse: Metrics and Models." ACM Computing Surveys 28(2), pp. 415-435, 1996.

Frakes, W.B. and Kyo Kang, (2005), "Software Reuse Research: Status and Future", IEEE Transactions on Software Engineering, 31(7), July, pp. 529-536.

<http://www.ganssle.com/articles/disaster.htm>

Knight, J. (2006). Faulty Human Communications: Its Impact on Dependability And What To Do About It., Presentation.

Carol Smidts, Xin Huang, James C. Widmaier: Producing reliable software: an experiment. Journal of Systems and Software 61(3): 213-224 (2002).