...Risk

For Programme Management / Project Management / Risk Management audiences

Introduction

Software Development is a Human endeavour... and humans make errors... the process of development is imperfect... therefore all software contains errors.

The number of residual errors (i.e. still present post development and its verification) has a number of factors including product architecture, process rigour, competency of the engineers, complexity of the product, effort expended.

In software, design, manufacture and commissioning are all part of a single process... the remaining processes are largely distribution, installation and operation which do not impart any variability.

Errors, Hazards and Risks in Software Systems

Errors can initially be considered dormant, they only become active (i.e. a Fault) when traversed. Not all faults may manifest in errant behaviour (i.e. precipitate Failure) depending on the robustness of the system.

Errors therefore are potential hazards... (in that they potentially become faults when reached)

1. Some may be 'unreachable' (system conditions can never be realised)
2. Some may rarely be traversed (exist in conditions which are rarely met)
3. Some may be avoided by operational procedure
4. Some may be created by unforeseen failure of the infrastructure (e.g. component failure)
5. Some may be created by malicious exploitation.

Risk is a product of the impact of the hazard (i.e. whether the Fault manifests itself as a Failure, and whether the defences are able to mitigate it) multiplied by its likelihood of occurrence.

Software is very deterministic... for the exact same set of conditions you can expect the same outcome! You might expect, therefore, that irrespective of how many times the hazard is met, if it is able to be mitigated by the defences, its impact is zero! Unfortunately the system response to that outcome is entirely dependent on the nature of the error and the state of the system. (E.g. an error that corrupts a critical memory location may be i) benign if the next system action is a valid update of that location, or ii) damaging if the next action is critical data read from that location).

Because of the high rate nature of software (millions or even billions of instructions a second), frequently occurring errors are readily identified, but some paths, scenarios or states (e.g. in exception handling or fault accommodation) are infrequently traversed, may be tested through limited scenarios, and errors may lay dormant.

Risk mitigation - Layered Responses and Defence in Depth

The 'double insulation' standard exploited by modern electrical power tools are a device to remove reliance on a single layer of protection from hazarding the user. The Rolls-Royce Specific Safety Policy for Control Systems found under RR GP PS1 requires that no control system is allowed to be only a single step away from a hazardous event.

Dependable systems rely on building layers of responses that are chosen to have few (preferably no) common mode failures (known as the “swiss-cheese model”), so that any error encountered may invoke a failure in that layer, but be isolated by another layer of the system such that the outcome is highly unlikely to manifest as a system failure. Individual weaknesses (imperfections) are represented as holes in slices of swiss cheese (individual safeguards or defences).

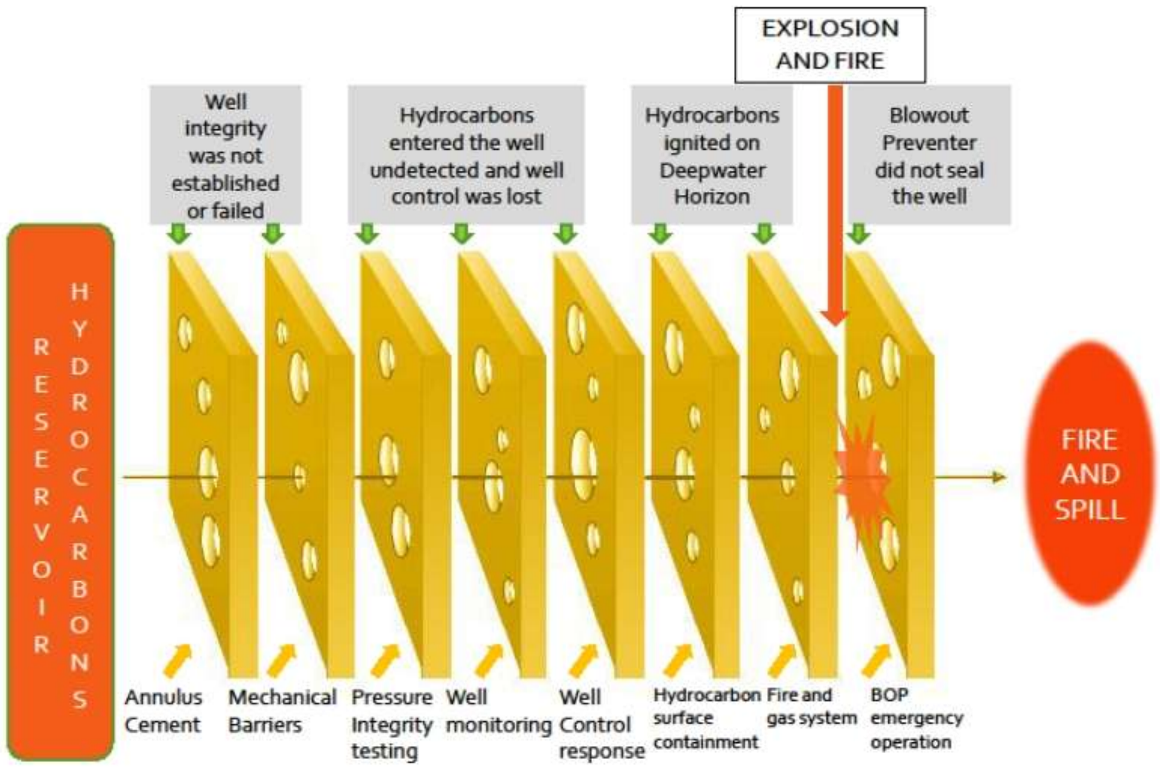
EXPLOSION AND FIRE
Well integrity was not established or failed
Hydrocarbons entered the well undetected and well control was lost
Hydrocarbons ignited on Deepwater Horizon
Blowout Preventer did not seal the well
RESERVOIR HYDROCARBONS
FIRE AND SPILL
Annulus Cement
Mechanical Barriers
Pressure Integrity testing
Well monitoring
Well Control response
Hydrocarbon surface containment
Fire and gas system
BOP emergency operation

Figure 1: An example Swiss Cheese model courtesy of the Bly Report on BP’s Deepwater Horizon rig failure.

Process rigour can decrease both the size and number of holes in a particular layer, by designing them out, or by investing effort in detecting and understanding their presence and ensuring some other part of the system provides a suitable defence..

Product architecture can help contain the impact of the hazard, through rapid detection of error and by minimise propagation of the error through a number of techniques such as minimising coupling, data flows, or defensive checks.

Competent system engineers can help ensure deliberate ‘misalignment’ of holes within individual layers and ensure sufficient differentiated layers to achieve the desired failure rate (“let-through”). This technique is often referred to as ‘defence in depth’.

The ‘holes-in-cheese’ is a reflection of the imperfection of our engineering endeavours against an idealised layer that would be entirely opaque (hole-free) representing complete proof against failure. The significant question remains that, if we were to model our defences in this abstract way, what the relative size of holes to integral areas of cheese might be!

The immeasurable risk - Humans in the System

Increasingly our systems include the human interaction as part of the system. System design can help 'manage' human operator interactions to improve avoidance of potentially risky areas, but human action is itself prone to error that needs to be designed out as far as reasonably practicable.

In some systems a human is left 'at the controls' for largely psychological reasons, but that potentially brings further risks. As automation increasingly takes away control of the mundane operation of a plant, so the operator's mental model of its operation and control fails to get reinforced. As a result he becomes less-practiced in recognising subtle indications and responding to them.

A human operator attempting to take control of a heavily automated plant, once the automation has resolved that it no longer can maintain control, is a potentially high-stress situation. What may have become heavily ingrained, almost instinctive, operator action for a complex plant may well be unpractised. (E.g. Compare a car driver of many years experience, regularly covering 60,000 miles a year, to someone who recently passed their test, who doesn't own a car but hires one for 2 weeks whilst on holiday!)

Worse still, it is likely that plant automation be layered in an attempt to fail gracefully, so the operator may find himself 'fighting' with the remnants of the automated control, for which he (the operator) has no valid mental model.

It is clear that the design of 'fail-safes', 'back-stops' and other 'emergency responses' that a human operator may engage has to suitably designed, have well-defined consequences, be clearly indicated and be free of complex sequences of operation, because of the deployment under duress.

Predictable software systems from less than deterministic electronic components

In an ideal world, we would build dependable systems solutions to high risk environments from a series of small, deterministic, fully validated components with good provenance, that we could integrate together and from which we could characterise the properties of the resultant system by analysis.

In software we are running instructions on a sub-micron level electronic device, with billions of individual cells, which have to work in perfect harmony, billions of times a second, with apparently random (or at least application dependent) sequences and patterns of use that the original manufacturer cannot forecast, whilst bombarded with events that potentially deluge the inputs of the system with noise, provide transient errors in the fabric and subject to potentially catastrophic failure with no recognisable or readily detectable precursor indication of degradation.

As we continue to miniaturise these systems, for ever increasing capability, improved power performance, reduced emissions, so the mechanisms of failure lose their dominant forms and we become more sensitive to a potentially frustrated system mix of sub-miniature processes that 'emerge' inexplicably, apparently aligned with use in individual applications.

These electronic systems become obsolete with increasing rapidity, being replaced with even more complex and potentially 'sensitive' systems, before sufficient time has elapsed to validate the fabric for its potential application, or of acquiring data that attests to its provenance. The rapidity of evolution of these electronic devices is driven by an insatiable consumer appetite for "smaller, faster, cheaper, less power hungry".

Any system that stays in service for a reasonable period, and exploits commercial components, provides the potential for an 'agent' to be introduced to exploit any weaknesses for malicious intent, or commercial advantage (e.g. causing safety incidents in order to damage reputation or manipulate share prices).

Although today we are focussed on safety in our risks, the increasingly software intensive product future and the potential connectivity it can bring, enables a significant world of security risk.

System security is usually a specific response to a defined threat. Software is ideal for this, in that it is easily adaptable and can therefore be readily changed to accommodate a changing threat scenario, particularly post original design. However, the use of software in a security system is itself a vulnerability... with 'ease of change' implying a potentially easier introduction of a malevolent agent throughout its life.

Increasingly Risky endeavours

As complexity increases, so our ability to foresee potential hazards decreases, the number of opportunities increases, our ability to detect them decreases, our ability to manage operators to understandable, safe and secure conditions decreases.

The system complexity that software systems maintain today is almost beyond belief. That we take for granted such complex systems for emergency communications (software defined radio, mobile phones), positioning (GPS) or even collision avoidance, emergency braking and vehicular stability, or voice recognition for many controls is astounding, given the relative rigour to which most of these systems were implemented, or the tools used to achieve their realisation.

Mechanical systems could not be realised at this scale of complexity, without realising the modern day equivalent of Babbage's problems with materials, precision, costs and physical constraints like friction.