Reflections on Resilience

Tom Anderson
Newcastle University, UK

Abstract

What do we want with regard to IT based systems (and networks of them)? We want:

- safety, for ourselves and others;
- security of information;
- systems that deliver *dependable* service.

And we want this from systems that are *resilient*:

- to the impact of change;
- to the inevitability of flaws;
- to the attacks of the wicked.

It's a very demanding requirement, and poses a tremendous challenge. This concluding lecture offers a few observations and hopes for the future.

Reflections on …
Requirements

Context: This is the final talk of the ReSIST Summer School on the gorgeous Mediterranean island of Porquerolles. For the past week participants have been grappling with the issues, challenges, techniques and technologies relevant to *Resilient Systems*. **Work and thought have never stopped since Monday.**

Derived requirements:

- Work is now over
- Forget the abstract, keep it light
- Finish early

Outline

Introductory stuff

Past, present, future

Reflections on:

terminology, concepts, wisdom,

consistency, simplicity, structure

Concluding stuff

Future systems …

Sometimes called *ubiquitous* systems:

- networks of networks of systems of systems
- heterogeneous, dynamic, evolving
- inevitably enormously complex
- multi-scaled (from nano-scale devices to server farms)
- ambient, with mobile access
- critical embedded infrastructures
- globally distributed

Future applications

Difficult to anticipate, but encompassing:

- personal communication, entertainment, enlightenment, enrichment
- transport
- commerce
- industry
- social and medical
- government
- military

Future characteristics

In contrast to (some) past experience, we need these systems to be

- Safe – not inflict death or injury
- Secure – not betray our secrets
- Reliable – deliver intended service
- Resilient:
  - Tolerant – to all manner of defects
  - Robust – to a host of attacks
  - Adaptive – to a variety of changes

In a word, we need them to be **Dependable**
– worthy of the trust and reliance we place upon them.

Reflections on …
Timing

Past: Dependability would be rather nice

- FTCS Symposia started in 1971
- Corrosive influence of IT failures

Present: Dependability is really essential

- Overdue recognition has, in part, been driven by the widespread impact of security weaknesses
- Framework 6 and 7
- NRC report on Software for Dependable Systems 2007
- Even Microsoft

Future: Dependability should be mandatory

Reflections on …
Terminology

Dependability

Voter

Defects

"Dependability" ?

Reliability, availability, integrity, security, safety, timeliness, confidentiality, privacy, trustworthiness, usability, maintainability …

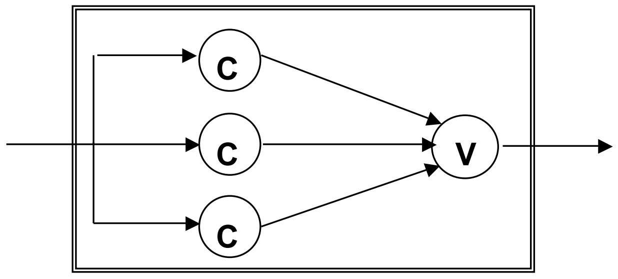
You name it …

But the systems we want should deliver on **all** relevant attributes.

Dependability was proposed as a label (signifier) to capture that one simple single generic attribute:

The system does all of the things we want it to do,
and doesn't do anything we don't want,
mostly.

The “Voter” ?



In the Triple Modular redundant system above the “voter” **V** is the element that ***doesn’t*** vote - it selects the majority decision from the elements **C**

“Defects” ?

Failures, errors, faults,
blemish, blunder, “boob”, breach, breakdown, bug, cock-up, crash, defects, deficiency, flaw, imperfection, inadequacy, lapse, omission, oversight, shortcoming, slip, solecism, weak point, …

When things are not as we would want them to be, there are only two fundamental concepts:

1. **Bad State** – the condition of the system is undesirable.
2. **Bad Event** – an undesirable occurrence in the system behaviour.

But whatever terminology you choose, there is no escape. All such situations are the result of a mistake or misjudgement that we made.

You can’t put the blame on the real world. It doesn’t care.

Reflections on …
Concepts

System Failure

A criterion of success (absence of failure) is imposed on the behaviour of the system. If we assert that before time *t* the criterion was always met, but that immediately after time *t* it was not met, then a failure of the system occurs at time *t* .

But all systems, hardware and software, always behave exactly as the laws of the Universe dictate. Either direct your complaint to the builder of the system or the imposer of the criterion. *Don't* blame the system.

Reflections on …
Words of Wisdom

Testing can show the presence,
but never the absence of faults.

[Edsger Dijkstra]

Safety is a system property.

[Everyone I ever met in the safety community]

Reflections on …
Consistency

A backward look at the origins of protocols for Byzantine agreement.

The "Interactive Consistency" problem.

Clock synchronisation in SIFT.

The SIFT project

Funded by NASA (Langley) in the late 70s.

An architecture to deliver a highly dependable avionics platform.

Prime contractor: SRI International in Menlo Park, California.

Multiple processing nodes; redundant computations; verified software to allocate computational tasks, coordinate the nodes and obtain a consensus result …

despite arbitrary node failures, up to a pre-specified limit.

The drifting clocks

SIFT coordination algorithms require the clocks in all non-faulty nodes to “agree” on the time.

But all clocks drift apart eventually, and therefore must be synchronised at intervals.

All non-faulty clocks must agree after synchronisation, even though some of the clocks may be arbitrarily defective (we have to set a limit on how many faulty clocks there are, of course).

The simplest non-trivial case is 3 clocks, at most one faulty.

The claim

“No such algorithm exists for 3 clocks.
There is an algorithm for 4 clocks, at most one faulty.
We have proved this.”

Lamport, Shostack and Pease (1979)

The arrogance of youth

A young researcher heard of the claimed proof, and was encouraged to seek for a counter example:
a solution algorithm for 3 clocks.

In just two days he had the algorithm: an elegant but utterly non-standard approach which *(a)* clearly worked and *(b)* exploited - for sure, he guessed - some unknown unstated assumption on which the proof assuredly depended.

Pride cometh before a fall

Young and cocky maybe, but not totally stupid. This was the plan.

Step 1: **Prove** new algorithm works.

Step 2: Send algorithm and proof to SRI. (Much hilarity.)

Step 3: Collect the rewards of fame and success.

Alas, two days (and nights) later the young researcher had abandoned algorithm and proof and instead had created from scratch a sketch of the SRI non-existence proof.

The insight

Classic approach for 3 clocks is the median algorithm: output all 3 values to all 3 clocks and synchronise on the median (middle) value.

But no assumptions (at all) can be made about the 1 faulty clock. It destroys the median algorithm by sending different values of its own time to the 2 good clocks.

The natural fix is to perform a cross-check on values to detect any inconsistent values from a suspect clock.

But none of this can ever succeed. The key point is that you must assume that the faulty clock knows **everything** that you do, however subtle. It's not only malicious, it is omniscient too.

The insight

The faulty clock was already aware of my method, whatever it was.

An epilogue

Just a few years later John Wensley published (in FTCS) a working algorithm for synchronising 3 clocks when at most 1 clock can exhibit arbitrary faulty behaviour.

His much better approach was to breach (quietly) one of the stated basic assumptions of the proof and thereby evade the argument. The interactive consistency community thought this was, well, rather evasive, but any practising engineer would regard it as an obvious way to proceed.

Reflections on … *Simplicity*

From Goethe to Eddington to Hoare

Everything is simpler than you think
and, at the same time, more complex than you imagine.
Goethe

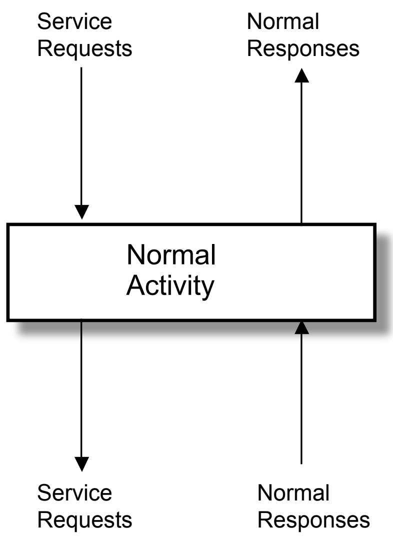
We used to think that if we knew one, we knew two,
because one and one are two.
We are finding that we must learn a great deal more about *and*.
Sir Arthur Eddington

The price of reliability is the pursuit of simplicity …
C.A.R. Hoare

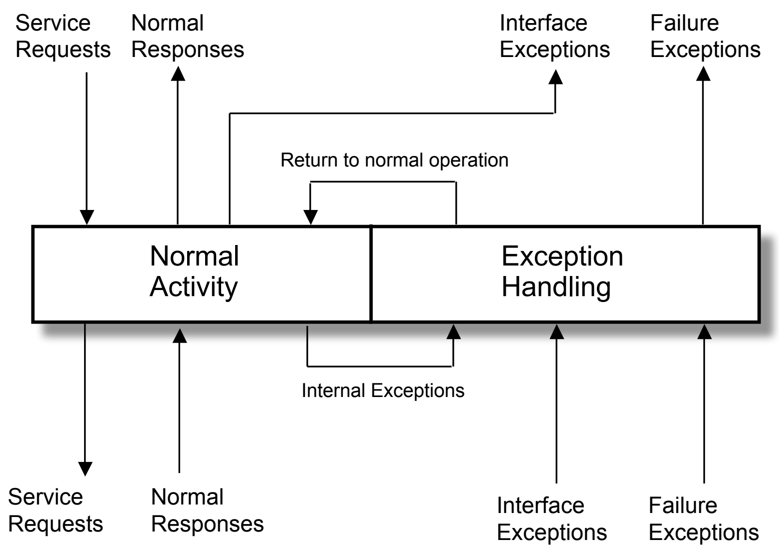
Reflections on …
Structure

If there is to be a generic "solution" to designing appallingly complex systems that are, nevertheless, dependable, then it must lie in underlying simplifying structural principles, which enable the salient and essential emergent properties of a system to be "guaranteed" despite the intrinsic difficulties.

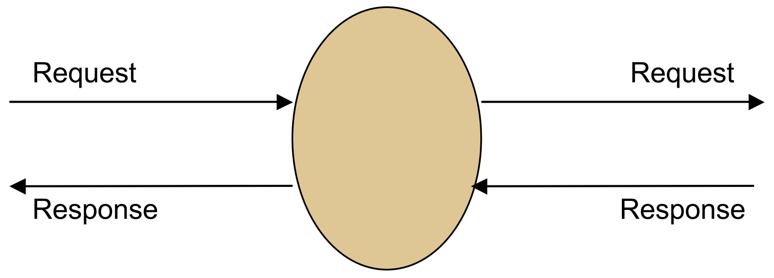
Basic module



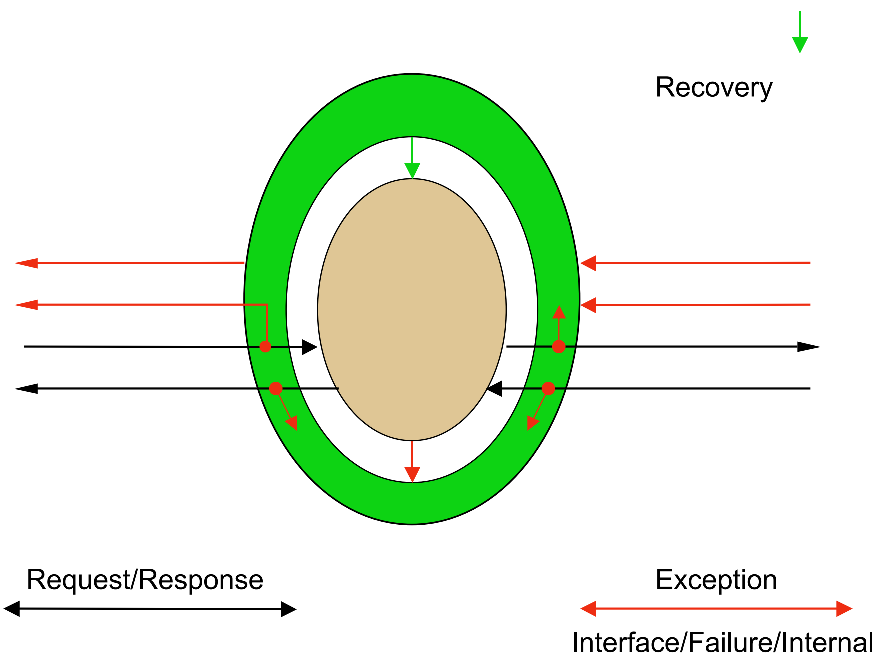
Resilient module (1981)



Basic process



Resilient process (2004)



Almost there …

A personal journey:

- from unaware logician [school]
- to mathematician in training [university]
- to inexperienced analyst [research associate]
- to aspiring engineer [professor]
- to despairing engineer [dean]
- to optimistic envisioner of rigorous socio-engineering of fantastic technical systems that are directly embedded in the society they serve (must read Asimov again)

Wrapping up

A personal view:

- System architecture must offer (almost) complete freedom
- System infrastructure must enforce fundamental constraints
    - Constraints can never be breached
    - All breaches are detected
    - Penalties are severe
- All elements should have a protective wrapper
    - Protection for imports and exports
    - Dependability (meta)data needs to be explicit and pervasive
- Resilience capability driven by wrapper response to metadata

Reflections on … *Programming*

Flon's Law

There is not now
and never will be
a programming language in which it is the least bit
difficult to write bad programs.
Larry Flon

Reflections

Sunrise on Mt Assiniboine, Canadian Rockies: Miles Hecker