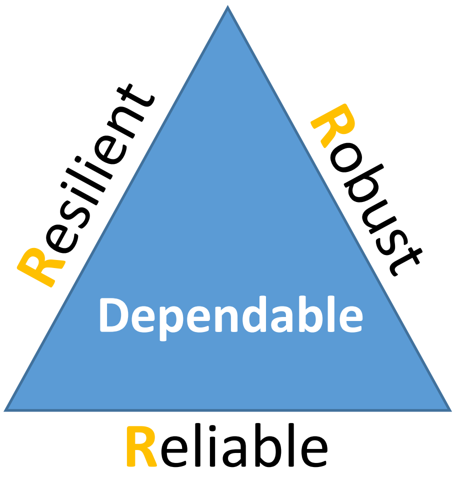
# The three R's of dependability



*Paul Fretter*

We need to feel that we can depend on computer services or systems, to a greater or lesser degree, according to their importance, exposure and development status. Almost by definition, critical systems need to be highly dependable, whereas faults or outages can be tolerated in non-essential and development systems. In this short article, I attempt to put into words what I think are the core components of the intuition or 'gut-feeling' that the seasoned sysadmin or service manager will possess. I propose the Dependability of a system can be described as a function of three properties: Reliability, Resilience and Robustness, specifically according to three simple questions.

| | |
|---|---|
| **Reliability** | To what extent can/will the system be **relied upon** **to behave consistently**, for a given set of initial conditions and inputs, and return consistently accurate and timely responses or outputs? |
| **Resilience** | To what extent can/will the system **continue to operate reliably**, and with little or no interruption, in the event of a hardware or software component failure? |
| **Robustness** | To what extent can/will the system **continue to operate reliably** under abnormal load conditions, or in the event of a spurious or malformed input/request, or even a malicious attack? |

If you bring one of your own systems to mind, and ask yourself these three basic questions, I suspect you will already have a reasonable feel for how dependable it is without looking any further. Whether you are planning a new system, or assessing something that already exists, the same questions can be applied.

To go into more depth, to convince yourself or to communicate to others, you can qualify your assessment further with a checklist of the factors you consider important to each question, and the table below illustrates a few examples.

**Dependability Checklist *(illustrative example)***

| Reliability | Resilience | Robustness |
|---|---|---|
| **Code and System:**<br>• Are the algorithms proven and is the code tested to give correct and consistent results?<br>• Change control.<br>• Are results or output delivered in a consistent timeframe?<br>• Is start-up behaviour consistent after un/controlled shutdown or restart.<br>• Interaction with other systems tested.<br><br>**Data:**<br>• Are routine data consistency checks performed?<br>• Data provenance and audit trail<br>• Data accuracy and relevance | **Hardware:**<br>• dual PSU, ECC RAM<br>• RAID, EC storage, hot spare HDD<br>• Failure detection and notification<br><br>**Code and System:**<br>• Failure detection and graceful handling<br>• HA clustering, VM migration<br>• Checkpoints and snapshots<br>• Data/system backup<br>• Handling failure of other interacting systems<br><br>**Network:**<br>• Multi-homed<br>• Diverse routes<br><br>**Environment:**<br>• Protected power - UPS and generator<br>• N+'n' cooling | **Hardware:**<br>• Clean, filtered power supply<br><br>**Code and System:**<br>• Input masking<br>• Handling unmapped requests or unanticipated state<br>• Patch maintenance<br>• Behaviour under heavy load<br>• Protect CPU, RAM and disk for core functions<br><br>**Security:**<br>• Authentication/Authorisation<br>• DoS, Intrusion detection<br>• OS hardening<br>• Data ACLs<br><br>**Network:**<br>• Intrusion prevention, ACLs |

This approach can help to highlight the basics quickly and give you a 'feeling' for how much effort is required to achieve an acceptable level of Dependability, and thus far with no mention of scoring or quantifying, making it simple to scope out a new project.

I want to leave it there, as a quick method of making a qualitative assessment.

To extend this into a formal quantitative assessment, it would be straightforward to devise a scoring system covering the factors within each question. In a future article, I will suggest ways to apply this.

Paul Fretter, August 2017