Improving Salience Retention and Identification in the Automated Filtering of Event Log Messages

by

Paul Radford

A thesis
submitted to the Victoria University of Wellington
in fulfilment of the
requirements for the degree of
Master of Science
in Computer Science.

Victoria University of Wellington
2011

Abstract

Event log messages are currently the only genuine interface through which computer systems administrators can effectively monitor their systems and assemble a mental perception of system state. The popularisation of the Internet and the accompanying meteoric growth of business-critical systems has resulted in an overwhelming volume of event log messages, channeled through mechanisms whose designers could not have envisaged the scale of the problem. Messages regarding intrusion detection, hardware status, operating system status changes, database tablespaces, and so on, are being produced at the rate of many gigabytes per day for a significant computing environment.

Filtering technologies have not been able to keep up. Most messages go unnoticed; no filtering whatsoever is performed on them, at least in part due to the difficulty of implementing and maintaining an effective filtering solution. The most commonly-deployed filtering alternatives rely on regular expressions to match pre-defined strings, with 100% accuracy, which can then become ineffective as the code base for the software producing the messages 'drifts' away from those strings. The exactness requirement means all possible failure scenarios must be accurately anticipated and their events catered for with regular expressions, in order to make full use of this technique.

Alternatives to regular expressions remain largely academic. Data mining, automated corpus construction, and neural networks, to name the highest-profile ones, only produce probabilistic results and are either difficult or impossible to alter in any deterministic way. Policies are therefore not supported under these alternatives.

This thesis explores a new architecture which utilises rich metadata in order to avoid the burden of message interpretation. The metadata itself is based on an intention to improve end-to-end communication and reduce ambiguity. A simple yet effective filtering scheme is also presented which filters log messages through a short and easily-customisable set of rules. With such an architecture, it is envisaged that systems administrators could significantly improve their awareness of their systems while avoiding many of the false-positives and -negatives which plague today's filtering solutions.

Contents

Acknowledgments

I would like to express my gratitude firstly to my supervisors, Andy Linton and Dr Ian Welch, for their knowledge, support and enthusiasm. They helped to distill my ideas into a useful form, tempered my rhetoric and provided necessary academic direction. I must thank the many former students of mine in Japan whose fortitude, simple humanity, and generosity inspired me to return to academic study in the first place - this thesis would never have been written without the Japan Exchange and Teaching (JET) Programme. Thanks also goes to the attendees of the New Zealand Network Operators' Group (NZNOG) conference of 2011, who withstood my presentation and then participated in my survey, the results of which form a very significant part of this thesis' payload. I am grateful to Roy Maxion of CMU for taking the time to advise us of a particularly pertinant piece of work which otherwise would have gone undiscovered. Finally, I would like to acknowledge the considerable support of my family and friends both before, and during, the period of my MSc study.

Terminology

The following terms are often used synonymously within this thesis:

- designer, programmer, developer
- program, (system) daemon
- (organisational) priority, importance

List of Figures

List of Tables

List of Algorithms

List of Listings

Chapter 1

Introduction

Computer systems administrators, as a part of their job function, must monitor event logs generated by their systems for signs of failure, impending failure, or security breaches. Many of these systems are simplistic in nature and produce well-defined output that can be easily filtered for important events. Many others, however, are inordinately complex, a situation increasingly common with the advent of multi-tier systems aimed at Internet commerce. This thesis will establish the position that this complexity is negatively affecting the ability of human operators to effectively monitor such systems, and that a fundamentally different approach is required to improve the situation.

The sheer volume of event log messages generated by today's Internet-connected systems is far beyond what was envisaged when system loggers were first conceptualised and implemented. It is entirely possible for one web portal system at a mid-sized university to generate at least 250MB of text in the form of log messages, every single day. The problem is akin to being assigned the task of checking the spelling and grammar in yet-to-be-published novels. This wouldn't be so bad if (1) much of the text wasn't replicated yet still superficially different, (2) you weren't being asked to check five novels per day, 24 hours a day, 7 days a week, 365 days a year, (3) the accepted spellings and grammars weren't changing on a daily basis, or (4) it wasn't possible that massive financial losses could occur, or even lives lost, if one single mistake made it through. The situation, though, is that all those four conditions can be true in the case of event log messages. And they're certainly not novels written for a general audience; the vast majority of possible messages are only able to be interpreted by the software author themselves, but we have no mechanism for definitively determining which ones those are. In fact, there is no existing way for any machine to genuinely decide whether or not a certain message really does warrant human attention within an organisational context. The result is that human systems administrators are bombarded daily with vast numbers of spurious messages that obscure the few ones of real value. These are the *salient* messages: those which humans would care about if only their attention could be properly brought to bear.

1.1 Salience in event log messages

Real-world computer systems administrators are professionals dealing most often with two broad categories of tasks: they have to contribute to, facilitate, or lead, the efforts involved with the implementation of new systems; and additionally they must maintain existing systems. The first focus for this thesis will be exposing the awareness an individual administrator must possess in regards to system state – such an awareness would include a working knowledge of the system components, as well as past, current, and potential future threats to the operational status of those parts. The only existing mechanism by which they can exercise such vigilance, other than point-in-time manual checks of resource usage (such as the disk space remaining, CPU load, etc.) is through monitoring of system events via event logs, or their synthesised, visual brethren in the form of 'dashboards' - as portrayed by Stephen Few in [14]. A measure of how well such monitoring works is the degree to which it assists the systems administrator in highlighting the salient events where a salient event is one that is prominent or useful in identifying potentially-abnormal system behaviour.

Using the open-source community as an example, the only widespread method of monitoring system event logs is that of filtering through regular expressions categorised into black- or white-lists: black-lists flag known-bad messages while white-lists allow innocuous ones to pass through. The remainder are typically e-mailed to the administrator as possibly-suspicious events. Anomaly-detection tools [15] using primarily-statistical routines also exist and have much academic work invested into them (e.g. [16, 17, 18]), but due to their relative rarity in actual deployments, the emphasis here will be on policy-based/filtering methods. The LogWatch [19] and Logcheck [20] projects typify this approach. Even though these capabilities exist and are free software, the Debian Linux Project [19] lists them as installed on only 3.2% and 3.9% of machines, respectively, from an 'opt-in' sample size of 91395 as of this writing [21]. On the commercial side, NMS [22], Snare Server [23] and Splunk [24] (among many others) also provide such functionality as part of their offerings.

The quality and consistency of input to any automated system (such as those mentioned above) is critical to the quality of its output, for it is here that the established log-monitoring solutions hit their first problem. In reality, systems administrators are called upon to upgrade software on a daily basis, even if only to implement fixes for identified

Listing 1.1 Selected examples of event log messages

```
Feb 3 12:35:56 gateway openvpn-server[2080]: 123.234.147.159:39936 TLS Error: TLS
    handshake failed
Feb 3 12:36:16 gateway openvpn-server[2080]: 123.234.147.159:40014 Authenticate/Decrypt
    packet error: packet HMAC authentication failed
Feb 3 12:36:16 gateway openvpn-server[2080]: 123.234.147.159:40014 TLS Error: incoming
    packet authentication failed from 123.234.147.159:40014
Feb 3 14:38:25 gateway kernel: ATM dev 0: error -110 fetching device status
Feb 3 14:38:44 gateway kernel: ATM dev 0: usbatm_complete: urb 0xdef9f2a0 failed (-84)!
Feb 3 15:12:32 server1 krb5kdc[4196]: TGS_REQ (3 etypes {16 1 3}) 192.168.1.100:
    PROCESS_TGS: authtime 0, <unknown client> for host/server1.example.com@EXAMPLE.COM,
    Request is a replay
Feb 4 13:06:42 gateway named[23870]: clients-per-query decreased to 10
```

security risks, and such changes impact on the effectiveness of an automated filter. An updated version of the popular Apache web server or of Microsoft Exchange™, for example, is likely to produce event log output which is not identical to the prior revision. With no notification of the problematic change provided to a busy administrator (changes to log messages are typically not reported in the 'changelog' list provided with software; such lists usually contain new features, configuration file format changes, and the like), a critical log message may no longer exactly match the "violation" black-list regular expression written to flag it. On the other hand, the white-list may not function correctly either, increasing the occurrence of distracting but legitimate messages in the daily report. It is not difficult to imagine the maintenance burden [10] incurred by major upgrades such as an entire operating system. Indeed, in this regard the particular technology used is practically irrelevant, whether it be data-mining, automated corpus construction or another alternative because, as re-iterated by Buckley & Siewiorek [25] in an extensive paper, analysis algorithms can only be as good as the quality of their input data. When that input data is modified and the filters are updated/trained/verified either at a later date, poorly, or not at all, the quality of the resulting (filtered) output is reduced.

This automated output, meant for administrators to review on a daily basis, suffers further from fundamental weaknesses stemming from the process by which event logs have evolved rather than having been designed, a lack of foresight by programmers as to how the logs would be used [26], and the almost-imperceptible fusing of human assumptions with genuine information in the output [25]. The vast and overwhelming number of messages alone is an intimidating challenge in many cases [27], even when visualisation is considered [28]. This is without the additional difficulties of parsing their hugely diverse range of message formats, types and schema [29] and performing the semantic reconciliation that is therefore necessary [30]. All of these factors are detrimental to a systems administrator's efforts towards understanding the current state of the systems under their control.

The most visible outcome is the regular monitoring e-mail, a selected sample of which can be seen in Listing 1.1. Such messages require an extreme level of domain expertise to decipher; without solutions available even from a modern search engine (the author has repeatedly attempted to find a definitive explanation of "clients-per-query...", without success). Coupled with the high rate of unpredictable change and the potential for new or unanticipated situations to produce never-before-seen messages, the existing arrangement serves both as a impeding factor for systems administration, and an intimidating barrier to entry for those not 'skilled in the art'. Salience is an attribute added by an experienced human to the messages; it is not readily encoded for us to extract any more than artistic merit can be mechanistically deduced from a painting. Art does not even require the extreme depths of domain expertise that log messages do in order to be appreciated, yet we persist with the log message 'interface' with its burdensome cognitive and memory workload, in the hope of cobbling together a mental perception of system state - which at its highest level of abstraction is either nominal or in fault to some degree.

1.2 Impediments for automated filtering

The cryptic natural-language nature of event log messages is not only a problem for systems administrators, but also for those designing automated filtering solutions. As mentioned above, regular expressions (or an equivalent technology) are the most common filtering technology in active use. Their complexity and syntax, however, makes them difficult to generate correctly - as Salzter & Schroeder said of access control mechanisms in 1975, if the user "must translate his image of his protection needs into a radically different specification language, he will make errors."[31, p.5]. Some examples of regular expressions (or *regexes*) used with the Logcheck [20] package under Debian Linux v5.0 can be seen in Listing 1.2. These regexes were included with the Debian OpenVPN [13] package - a GPL-licensed, open source Virtual Private Network (VPN) service - in order to extend Logcheck's capabilities, and are from the white-list, meaning that all matching messages will be discarded, irrespective of their quantity.

Note that binary matching with a regular expression is an out-of-context, line-by-line mechanistic process which does not take into account any surrounding influences which might increase the salience of an otherwise unimportant message. Such filtering mechanisms obtain one and only one piece of information from a regular expression: it either matches *or* does not match a given line of event log output. A machine cannot determine the category to which any given message should belong, nor can it cope with unanticipated situations or messages. When the regular expressions for message matching are maintained separately from the codebase generating those very messages, keeping the two

Listing 1.2 Examples of Logcheck rules (regular expressions) used for event log message filtering [13]

```
^\w{3} [ :0-9]{11} [._[:alnum:]-]+ (openvpn|ovpn-[._[:alnum:]-]+)\[[0-9]+\]: (Outgo|Incom
    )ing Control Channel Authentication: Using [[:digit:]]+ bit message hash '(SHA1|MD5)'
     for HMAC authentication$
^\w{3} [ :0-9]{11} [._[:alnum:]-]+ (openvpn|ovpn-[._[:alnum:]-]+)\[[0-9]+\]:( ([-_.[:
    alnum:]]+/)?[.[:digit:]]{7,15}:
[[:digit:]]{2,5})? Connection reset, restarting \[[[:digit:]]+\]$
^\w{3} [ :0-9]{11} [._[:alnum:]-]+ (openvpn|ovpn-[._[:alnum:]-]+)\[[0-9]+\]:( ([-_.[:
    alnum:]]+/)?[.[:digit:]]{7,15}:
[[:digit:]]{2,5})? (Data|Control) Channel MTU parms \[[[:upper:]:0-9/ ]+\]$
^\w{3} [ :0-9]{11} [._[:alnum:]-]+ (openvpn|ovpn-[._[:alnum:]-]+)\[[0-9]+\]:( ([-_.[:
    alnum:]]+/)?[.[:digit:]]{7,15}:
[[:digit:]]{2,5})? TLS Error: Unknown data channel key ID or IP address received from
    [0-9.]{7,15}:[0-9]+: [0-9]+ \(see FAQ for more info on this error\)$
^\w{3} [ :0-9]{11} [._[:alnum:]-]+ (openvpn|ovpn-[._[:alnum:]-]+)\[[0-9]+\]:( ([-_.[:
    alnum:]]+/)?[.[:digit:]]{7,15}:
[[:digit:]]{2,5})? TLS Error: local/remote TLS keys are out of sync: [0-9.]{7,15}:[0-9]+
    \[1\]$
```

synchronised is a formidable challenge in itself, and one which (in this author's experience with these tools) has not come close to being achieved. Furthermore, the large number of unhandled messages prevents any meaningful statistical analysis of handled-message counts due to incomplete information; given that the unhandled messages likely contain relevant events which haven't been handled, simply due to the limitations of the filtering mechanism itself.

One must also consider that an inherent 'lag' exists in *any* solution which requires (or even benefits from) maintenance or training, in that any new/unhandled message which passes through the filter must be either manually added to the filter, or achieve enough statistical significance to warrant exclusion by a data-mining or machine-learning algorithm. In a sense, even manual addition is the result of a form of statistical significance, as a message will usually only be added by a systems administrator once its occurrences pass an arbitrary 'annoyance' threshold. Thus, the unhandled messages which are e-mailed on a daily basis are the most common output of such tools.

The bad-event detection scenario presents a danger, too, in that any reliance on black-lists (i.e. attack signatures) by virtue of relying on an exact match to those signatures can result in a false sense of security when no match is found. A lack of usable metadata has lead to results which are not reached by an evaluation of whether the message is important or not, but rather by static string matches which drift away from commonality with the codebase over time. It is therefore obvious that a never-before-seen message is considered on a basis of newness rather than whether a human needs to see it; all unanticipated events are presented as equals, irrespective of actual priority. Current systems are severely limited by the natural language content of these messages; there is no useful, easily-accessible metadata that can read both by machines and humans.

1.3 Research criteria for an improved alternative

Metadata that is machine-readable offers the tantalising possibility of deterministic filtering without the need to actively interpret and make poorly-informed decisions regarding the input data. While the input data should ideally be communicating the programmer's or program-designer's well-informed thoughts on the event in question, and be doing so in an effective manner, we only receive truncated and barely-descriptive messages such as those displayed in Figure 1.1 - crippled communication by any measure if we consider the saying “a picture is worth a thousand words”.

The metadata also has to survive the filtering process itself - even if only to inform that process. When we accept as a premise that the systems administrator needs the latitude to adjust their filtering solution, it becomes clear that the metadata has to be trivially simple to view; not obscured in a binary bit-packed form - as RFC 5424's “severity” field [11] currently is. Tuning a system for “better” performance (performance being regarded here as the quality or usefulness of the output) naturally requires knowledge of the input to that system, and in the case of filtering event log messages based on metadata, that input is the metadata.

Furthermore, there is a limitation currently present in event log messaging systems which may or or may not have been obvious in the examples above: the difficulty (some might say “impossibility”) of specifying policies that accurately reflect the concerns of a person and/or their organisation. Incorporating these realities with regular expressions requires a duplication and line-by-line customisation of the rule base for each object (e.g. application daemon, physical server, etc). Data mining algorithms face enough challenges as it is, with the filtering of log messages against policy, let alone the burden of filtering with gradiated responses, explicit confidence levels and varying methods of communication. Neural networks present no details of their internal workings; one would presumably require a separate network for each policy, and their results would likely be inconsistent.

The metadata, in whichever form it takes, therefore needs to facilitate the automated application of real-world policies, so that the capabilities of this alternative filtering approach prove sufficiently advantageous to justify the effort involved in its implementation. Policies are not supported by any of the log message related literature reviewed in Chapter 2; their only appearance is in heavily-instrumented industrial plant monitoring solutions.

1.4 Potential improvements to software systems

If the problems above can be dealt with, the implication is that a result from this work would be software which fulfills the following aims:

- reduced duplication-of-effort
- deterministic and reproducible output
- minimised maintenance requirement in normal operation
- customisable policies that manipulate the output according to personal/organisational preferences and support multiple modes of communication determined by an event's real-world priority
- improved communication, with greatly reduced ambiguity, from programmer/program designer to end-user/systems administrator.

All these aims are held in contrast with the current situation in event log message filtering. The vast majority of log messages are never observed, either by humans or any automated process, and the automated tools that are employed entail their own substantial maintenance and implementation burdens. No end-to-end approach yet encountered has ever attempted to improve the source data of the log messages in question - prior efforts such as the Syslog protocol documented in RFC 5424 [11] have instead focussed on message routing and storage. As will be examined in detail, the entire architecture of logging and filtering, as it stands today, requires deep reconsideration in order to improve the monitoring of computing systems which are having to cope with ever-increasing challenges.

Chapter 2

Literature review

> “`lp1 on fire (One of the more obfuscated kernel messages)`”
>
> – *fortune* package from Debian Linux v5.0

The amount of literature on the specific topic of event log system design is scant at best; only two papers could be located which pertained directly to the problem at hand, and both lamented this scarcity [25, 32]. Buckley & Siewiorek performed an in-depth analysis of the event logs produced by VAX/VMS systems and found that even a tightly-controlled platform/codebase managed by one entity (Digital Equipment Corporation, in this case) suffered from inconsistencies and generally lower-than-expected quality in the event logs, despite being one of the best examples of its day [25]. They witnessed a self-centred design philosophy wherein the only non-end-user output from the system or system daemons was in the form of event log messages, yet the programmers writing the code that issued the messages were doing so exclusively for their own benefit. Debugging code remained in shipping products and polluted the event logs. Genuine events were populated with undocumented, personally-assigned codes. The systems administrator was the intended user of the product (with non-technical purposes catered for with their own applications abstracted from the OS) yet was not factored into any interface decisions beyond the command-line shell. In short, the programmers/designers appeared to be demanding the administrator possess their level of domain knowledge.

Etalle, Massacci & Yautsiukhin took a security/privacy-oriented look at the landscape; their framework allows one to formally ascertain the qualifications of an actual system, weighing up audit integrity and privacy. Unfortunately, some of their suggestions exemplified a ‘security through obscurity’ mindset, such as denying access to the `/etc/passwd` file in a Linux system with the goal of hiding usernames and therefore preserving privacy [32]. Etalle et al. had a focus on evaluating existing or proposed systems with a temporal bent; their criteria included such factors as “partial completeness”, “context independence”, “chaining” and “exactness”. Their efforts to ’straddle the fence’ on privacy-vs-usefulness in log files revolved around disposable pseudonyms for the most part. In any case, their research did not consider usability for the systems administrator or actual selection of one or a set of messages from a set of thousands or millions.

Simultaneously, however, there is substantial literature dealing with the formalisation of the analytical side of logging/event handling systems [5, 33, 17, 34]; such papers most often attempt to reduce false-positive/negative rates by applying improved data-mining techniques. Their outcomes are typically characterised as ’probabilistic’ and admit that false-positives and -negatives will occur; effort continues in this direction and a breakthrough cannot be ruled out. On the other hand, the task these researchers have set themselves boils down to constructing a strong artificial intelligence, in that they aim to convincingly interpret natural language (i.e. the text contained in the log messages) using experience, such that novel information can also be handled as though human intuition were present. The well-known and rocky history of AI suggests that such success may be either in the distant future, or perhaps impossible with current or even currently-foreseeable technology.

Other efforts have emerged with differing priorities and design emphases, which led to results satisfying other goals, such as a logging format for digital libraries using XML (e.g. [35] and to a large extent [36]), encrypted yet searchable logs (possibly utilising pseudonyms as well) that deal with the issue of privacy during audits (e.g. [37, 38, 39]) and the statistical analysis of alarms via machine-learning from a trouble-ticket database [10]. Entire grammars have been proposed, ranging from the building of parsers for audit logs [40], to the GEM language for coding and dealing with the log messages themselves [41]. GEM is an interesting case in that the authors’ examples appear to rely on a level of instrumentation that simply has never been present for most events occurring in distributed systems. GEM also seems to require flawless parsing of messages and equally superb anticipation of scenarios, in order to reach its true potential. Nonetheless it represents a thoughtful effort that could be of use with higher-quality event log messages as an input.

What can be readily observed is that the vast majority of literature related to the topic of event logging accepts the current state of poor and inconsistent input from system processes. Buckley [26] in his PhD thesis, and when writing with Siewiorek [25], appears to be practically alone in his calls for higher quality event log messages. Instead, attempts to enhance the quality of the analysed/filtered output via the well-established academic doctrine of incremental, iterative improvement, appear to be the academic 'norm'. These enhancement efforts, including those cited in the paragraphs above, typically come in the form of new or revised analysis algorithms (i.e. entirely post-collection improvement), ideas for corpus construction (i.e. improving analysis through the incorporation of another vetted input, as in [42]), or architectures/frameworks - either pre-collection (e.g. [36]) or formally assessing a logging system after the fact (e.g. [32, 40]).

What appears to be lacking in the literature is any examination of event logging from a usability perspective, other than when the topic was brushed against by Buckley & Siewiorek [25]. Event logs are apparently accepted as-is by authors, without critical analysis as to their suitability for purpose. In particular, the question that has not been considered is: "how do we deterministically select the 'necessary' messages, when we don't even know the criteria for which ones are necessary?"

2.1 Discovering "salience"

Applied models of salience in computer science are thin on the ground, and none could be located which were directly relevant to one-line event messages. Salience as a concept seems to be limited to the pure HCI domain; it has been explored in relation to games and graphical interfaces, but sadly not in textual analysis. The closest equivalent in logging systems is the notion of 'severity' which is encapsulated in the de-facto standard RFC 5424 [11], or perhaps the 'perceived severity' of the ITU's X.733 standard [43]. Severity by itself has become limited to a very local scope (that of the issuing program itself); this illustrates the point-of-view of the programmer or designer of the program. Indeed, RFC 5424 accepts that "severities are very subjective, a relay or collector should not assume that all originators have the same definition of severity" [11, pg. 36], reinforcing the impression that event log messages are written for the programmer/designer rather than effective, i.e. calibrated, communication to another audience.

The severity levels are shown in Table 2.1. In practice, they are conveyed in a 'priority' value which is a bitwise combination of the severity and 'facility' values - facility being the number used to route the message into a particular destination file - and the priority is discarded (by default) after message routing has been completed [11]. Newer software such as Rsyslog [44] is capable of displaying the severity value but again this is not the default behaviour. The RFC 5424 approach is to consider the *type of output* (i.e. the facility number which is intended to categorise messages into authentication, e-mail, CRON, etc files) and the *severity* as (together) indicating an absolute priority.

Given the well-known UNIX security/ring-domain approach, from a systems point of view

Numerical code	Severity
0	Emergency: system is unusable
1	Alert: action must be taken immediately
2	Critical: critical conditions
3	Error: error conditions
4	Warning: warning conditions
5	Notice: normal but significant condition
6	Informational: informational messages
7	Debug: debug-level messages

Table 2.1: Syslog severity levels as defined in RFC 5424 [11]

only the kernel could ever issue a severity "0" message. Other programs, however, would certainly consider some of their events to be "emergencies", such as a daemon monitoring a pool of redundant disks for failures. In terms of the severity levels forming a linear-numeric scale, each program tends to align the extremes of the scale with the limits of its own scope, regardless of whether it is a mission-critical database or a frivolous chess-by-email service.

The X.733 "perceived severity" concept [43], however, gets closer to a notion of salience. It includes the idea of user perception. Many event messages issued (e-mail server daemons are a good example) may be consequential to the systems administrator (e.g. the failure of an indexing mechanism meant to improve scalability) but entirely inconsequential to a user whose only concern is whether or not their e-mail appears in a timely manner. Even perceived severity, though, is entirely service-oriented and built from a one-size-fits-all perspective. It is also reliant on a static list of probable causes which are technology-specific and therefore immediately outdated. X.733 was finalised in 1992 and the list includes items such as "multiplexers" and "DCE-DTE Interface Error" [43, pg. 15] which may be of limited utility as of this writing.

A NASA paper [45] by Schreckenghost et al., that was primarily concerned with HCI issues for astronauts, made significant mentions of 'notification saliency' but never offered a formal definition. Schreckenghost et al. also relied heavily on a static ontology (similar in nature to X733) as well as information unavailable to an event logging system, such as a person's physical location, daily schedule, and so on.

Game theory is perhaps the "next best" example of an area of study concerned with individual events that are part of a greater whole. Andrew Colman's article on salience in pure-coordination games [46] points out that humans are intuitively drawn to perceived common touchstones based on knowledge of shared cultural understanding, among other factors. While this represents an insight into binary decision making in a competition, it also shows that an implied cultural context is necessary for such systems to be balanced.

The nature of this context is meant to be shared, but unspoken, by all concerned - meaning that salience can be viewed (in game theory, at least) as predominantly a human factor overlaid onto an otherwise rational game of pure chance. As Colman explains, the fact is that British and American players of "heads or tails?" choose "heads" 87% and 86% of the time, respectively, regardless of the even probability of outcome. That they regard "heads" as a "salient focusing point" is a given, but the exact reasons are not yet adequately known [46].

A more fine-grained scale can be found in the "C Language Integrated Production System" (CLIPS) - an expert system framework by Gary Riley - in the form of its "salience rule property" [47]. CLIPS can interpret a per-rule salience value as a run-time ranking order, meaning that rules with higher salience will be executed before rules with lower salience. The scale is given as a range of -10000 to +10000, with a default of zero being assigned if salience is left undefined, and this is meant to be interpreted by the person implementing CLIPS as a *priority* indicator [47]. Its utility is obvious in the context of an expert system. An expert system, at least in 1991 when CLIPS was made available, is supposed to present a linear series of questions to a user in order to aid them in root-cause discovery, decision making, or the like.

An unambiguous method of ranking is therefore advantageous when considering such a one-dimensional (i.e. sequence in time) interface presentation. As discussed above in regards to the RFC 5424 syslog severity scale, the concepts of 'severity' and 'priority' have in the past been combined, and the CLIPS salience value appears to be something which could more properly be termed as a *priority* value, given that it is referred to as such in documentation, and used exclusively for ranking. The nature of such ranking schemes (as in CLIPS) implies that no two alternatives should have an identical priority. This clashes somewhat with the human notion of salience; one of the more difficult moments faced in reality by a systems administrator is when two ore more items are competing for their attention but choosing between them is impossible - i.e. the items' salience cannot be adequately distinguished.

2.1.1 A further exploration of 'priority'

Prioritisation, though, was the primary focus of an extensive article by Wallin, Leijon and Landén. Their stated aim was to automatically prioritise the multiple hundreds of thousands of alarms which were daily swamping a telecommunications network operation centre (NOC) [10]. Like event log messages, these alarms had a very low signal-to-noise ratio. Very high levels of domain expertise were required, leading to a heavy and repetitive workload for the few so endowed. The alarms were also heavily standardised and therefore quite static in their presentation - a point of difference with event log messages which are often free-form text and prone to un-notified change - Wallin et al.'s "additional text field" contained a mere 3500 unique values [10, pg. 9].

Log messages have the further distinction of *not* retaining or indicating state, which *is* a common feature of the alarms used by Wallin et al. Their approach was to clean the data flow (consisting of alarms and trouble tickets), turn a data-mining algorithm loose on the cleaned data set, and feed the results to a neural network, in the hope of creating a hybrid filtering/expert-system which would utilise information from the trouble-ticket database to assign correct priorities to new alarms.

Wallin et al., as part of their data analysis, identified that the correlation between alarm 'severity' and manually-assigned 'priority' (by their definition of priority as used in the telecommunications NOC) was very weak, leading them to conclude that "severities cannot be used as priorities" [10, pg. 14]. This is a key point to remember with regard to vocabulary overloading: "priority" is often used in several different senses. In RFC 5424 it refers to severity+facility; in X.733 as more of a "perceived severity"; and in expert systems and Wallin et al. as ranking data. What Wallin et al. are pointing out is the mismatch caused by the programmer/designer deciding upon severity within the context/scope of the program's functionality whereas the systems/network administrator assigns priority based upon a wider context that considers the program's role in terms of achieving organisational goals[1]. While there is undoubted value in each different interpretation, they cannot and should not be equated, as is so often done.

The conclusion reached by Wallin et al. was that their neural network could correctly assign priorities to 50% of incoming alarms. This is an improvement over their "naïve approach" with its 17% success rate [10]. It does, however, leave 50% *incorrectly* classified - meaning none of its output could be trusted by humans and would inevitably be disregarded. Note that a human expert was required to determine the percentage of correct vs incorrect.

The neural network approach also relies on the trouble-ticket database, leaving it unequipped to deal with novel situations, such as those that might be encountered with a router OS upgrade, a routing topology change, or a malicious network attack like a distributed denial-of-service. New issues must achieve statistical significance before any data-miner will regard them as anything other than spurious noise, then be learned by the neural network; leaving a considerable time lag in which humans must deal with the

[1] The programmer/designer point-of-view includes severity data: i.e. a narrower focus on the program being worked on; product/code context. The systems/network administrator point-of-view on the other hand includes priority data: i.e. a broader focus on the organisation they belong to, end-to-end Service Level Agreements, etc; organisational context.

unfiltered alarms as they do now, *and* create and solve the related trouble tickets (in Wallin et al.'s NOC), all with no deterministic reassurance that their efforts will ever result in correct classification. If the neural network does begin to classify the new alarms correctly, it cannot even provide a notification of doing so. Such a system is the archetypal 'black box'.

2.1.2 The psychological point of view

This story, of over-reliance on terms such as severity or priority, leaves those of the humanist philosophy distinctly unsatisfied. It would surely be folly to entirely ignore the large body of related psychological research. J. Richard Eiser, in a 1971 article on individuals' perceptions, mentions that the "concept of salience, like the concept of a stimulus, is more easily employed than defined." [48, pg. 444] Eiser's primary investigation was into the correlation between the strength with which an opinion is held, and the increasing extremity of that opinion.

That is, an opinion or reaction which is more forceful in its expression will tend to be an opinion or reaction which is further away from the 'middle ground' of society's norms, i.e. towards either end of a response/survey dimension. In the process of his research, Eiser had to formulate a concept of "dimensional salience" - and concluded that the various dimensions that are found to be relevant to a given conceptual idea will be evaluated differently by individuals. In other words, the dimensions themselves possess a salience attribute; "[t]he individual's frame of reference must therefore be defined in terms of the dimension or dimensions he regards as most important or 'salient'."[48, pg. 443]

We could perhaps therefore benefit by evaluating the salience of dimensions, rather than simply accepting dimensions as though they were set in stone. The declaration of a dimension (for example, severity, or priority) as the single measure of how salient an event log message actually is, therefore flies in the face of Eiser's recommendation. It may very well be that program designers, or programmers, generally find the concept of severity fits their purposes; that it makes doing their jobs that little bit easier. Eiser queries: "...which dimensions will be salient to a particular individual?" [48, pg. 443] but it appears that question has not yet been asked with regard to event log messages, Syslog standards, or the like. Systems administrators have to suffer the consequences of such declarations and appear to have minimal, if any, ability to specify how well their primary interface into a system's inner workings actually functions.

Certainly it is clear that one dimension will not fit all individuals when it comes to a concept as amorphous as salience. An emerging theory belonging to management psychology, "stakeholder salience", seeks to establish the idea that there are several dimensions which managers use to judge how they should allocate their finite time when dealing with stakeholders in their businesses. Systems administrators, too, have to make very similar value judgements on a daily basis, especially in the more outsourced environments. A very widely-cited 1997 article by Mitchell, Agle and Wood [1] offers a logical process of deduction for the three dimensions they select for consideration: 'power', 'legitimacy', and 'urgency'. All three are optional, i.e. not all stakeholders will possess all three. They use a Venn diagram, shown in Figure 2.1, to illustrate the concept; the 'most salient' stakeholders possess all three attributes and therefore reside in the central union.

Mitchell et al. proposed these three dimensions in an attempt to delineate the factors people use in their evaluations of claims on their time - the multi-dimensional scale is a response to their premise of no one attribute being reliable enough to capture the complexity inherent in "identifying stakeholders, as well as in the agency, behavioral, ecological, institutional, resource dependence, and transactional cost theories of the firm." [1, pg. 854]

Mitchell et al. began with Freeman's broad definition of a stakeholder: "any group or individual who can affect or is affected by the achievement of the organization's objectives" [1, pg. 854], which they acknowledge as so broad as to not exclude any potential candidates. Such a definition finds much in common (in scope, at least) with my own working definition of message salience. It should be emphasised here, as Mitchell et al. themselves emphasise, that the idea of stakeholder salience is not to provide advice or a framework to deal with future situations, but to reliably identify what factors are used in reality as people make decisions.

The notion of 'power' is the ability for one stakeholder to change outcomes in favour of their own preferences, i.e. "to impose their will" [1, pg. 865]. It revolves around coercion, regardless of the tactics used to achieve that coercion. In scope, 'power' appears to be limited to individual parties in a relationship, i.e. few actors, as power could be said to become increasingly diffuse as more actors are involved. 'Legitimacy' represents what some might say is the other side of power's coin: it is the degree to which an action might be considered acceptable in a larger context, and is judged according to the perceptions of others [1]. Consequently, those who consistently violate the norms of an organisation or society-at-large tend to find their legitimacy weakened to some degree or another. Furthermore, Mitchell et al. recall Weber in using the term 'authority' to communicate "the legitimate use of power".

Whereas power and legitimacy are distinct, yet inter-related, 'urgency' finds itself internally cleaved in two: the ideas of criticality/importance and time-sensitivity encompass

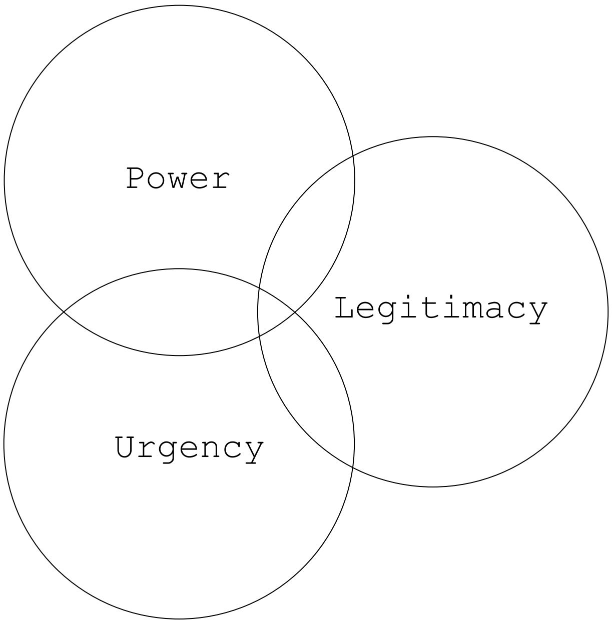


Figure 2.1: Dimensions influential to management decisions [1] (numerals removed)

(potential) exposure with the entailing risk, and the pressures of timeliness/deadlines, respectively [1]. That is, for a stakeholder's demands to be considered urgent, they must possess both those features. "Validity" is perhaps another word that can clarify this position; the stakeholder's stated requirement of immediate action cannot be valid without a truly critical situation *and* genuine time pressure. It is not a large leap to the position that 'urgency' is the most granular of the dimensions - while power and legitimacy (possibly combined as 'authority') tend to be amorphous and difficult/impossible to nail down to particular action items, 'urgency' is often tied to a specific goal or requirement. In summary. power and legitimacy are most often thought of as personal attributes; 'urgency' on the other hand is attached to outcomes. Taken together, these dimensions form a measure of salience for stakeholder claims [ibid].

The idea of the three dimensions for management is Mitchell et al.'s response to their perception of scholarly calls for stakeholder theory to "articulate a normative core" [1, pg. 882], which they define as the search for reasons why "some claims and some relationships are legitimate and worthy of management attention and why others are not." [ibid]. Event log messages require similar investigation, but unlike Mitchell et al. where human nature was being looked at, such an investigation would have to juxtapose human nature on the one hand and chaotic log messages on the other - considering the junction where the two interact.

2.1.3 A successful measure developed in medicine

The discovery of a three-dimensional scale for ennumerating demands on one's time recalled a system used in medicine, structured along similar lines. The modern triage process in hospitals relies in part on the Glasgow Coma Scale (GCS): a set of three dimensions for assessing and communicating a patient's neurological condition [49]. Medical professionals have embraced the scale since its debut in 1974 [12]. It ennumerates the motor responses, verbal capability, and eye-opening characteristics of a patient; optionally these numbers can be combined into one, ranging from zero (i.e. dead) to 15 (normal healthy adult).

Although the GCS is not without its weaknesses, mostly related to the practice of adding the dimensions together [12], it deals neatly with the clinical need to dependably communicate important information from one human to another. The GCS score presented by a paramedic upon delivering a new patient may be written on a chart and later referred to by doctors and nurses to judge the patient's improvement or decline since presentation. Departments within a hospital, and across hospitals, all rely on the same scale.

A low GCS score automatically raises the patient's priority for receiving treatment - a critical part of the triage process, especially in a busy hospital. Every medical institution is limited to a finite set of resources and must apportion those resources according to genuine need. This need usually cannot be assessed by the patient themselves; after all, most people would assign a higher-than-necessary priority to their own situation because they are considering a local, self-centred scope (much like the earlier concept of severity). The "correct" priority can only be assigned by an individual with knowledge of the organisational situation: load vs capability, as well as an absolute assessment of the problem based on experience of the breadth of possibilities, from the most trivial to the worst-case.

In essence, the GCS assigns a salience value (from the point of view of the medical professionals) to each individual patient. It is worth noting that the range of diagnoses it covers has changed only slightly since 1974, and that it is considered a discriminative (between causes), predictive (of outcome) and evaluative (of present condition) scale [12]. The GCS has been thoroughly tested in the medical environment and is now generally considered indispensible.

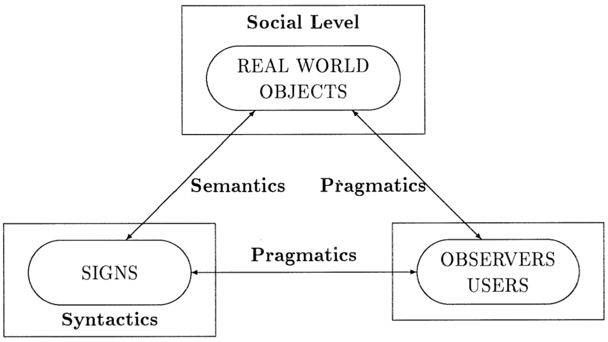


Figure 2.2: The 'representational triangle' [2, 3]

2.2 Commonality of meaning

As identified by Prasad, the GCS actually scores highly in terms of common understanding, with the primary discriminator being experience with its use [12]. Given the concrete nature of human medical treatment and common, undoubted motivation among those involved, such a result is not surprising. This commonality of meaning, though, is by no means assured simply by the use of scales, whether single- or multi-dimensional [50]. Scale response is fundamentally a categorisation exercise and it has been posited that we can never prove identical efficacy across individuals simply because "the absolute meanings of the response categories are unknown." [50, pg. 244] As Cook & Campbell also remind us, experimentation in this (human) domain must be acknowledged as being at the mercy of innumerable unknown and unaccounted-for factors [51], one of which is simply "meaning" - always communicated indirectly through the use of "signs". The very idea of a common meaning is imperiled in event log messages because of their frequently absurd complexity, with few common signs.

2.2.1 Semiotics: the study of signs

The use of semiotics in computer science has generally been limited to the human-computer interaction (HCI) area, or attempts to deal with terminology. An example of the latter would be Barron et al.'s article [2] on terminology used to identify and classify various breeds of information system, such as 'decision support system', 'expert system', and so on. Such jargon can represent an almost-impenetrable professional dialect, much like that used by doctors in concert with the Glasgow Coma Scale discussed above. Barron et al. draw on Stamper's [3] work on semiotics in representing the context around terminology with a triangle (Figure 2.2) which dates back to Charles Peirce's seminal work in the early 20th century .

Accepted semiotics theory elucidates that signs are abstract terms representing various

Feature	Basis in semiotics
Application domain	Social level
Action complexity	
Social consequence	
Acquisition complexity	Pragmatics
Acquisition scope	
Input usability	
Output usability	
Justification	
Real world relationship	Semantics
Representation	Syntactics

Table 2.2: “10 features of every Information System” [2]

aspects of real-world objects and social norms. That is, a sign has no meaning without knowledge of what it represents. Additionally, signs not only signify things or actions in the real world, but rely entirely on the “social reality” [2] that is understood between individuals and groups of individuals. One could perhaps characterise signs as labels for discrete chunks of context. That context is not static – the use of a sign can influence it, after all – but instead is part of the ever-changing present and future of our world. When the topic of this thesis is applied to the representational triangle, the Observers/Users are the systems administrators; the Real World Objects are the machines and programs which are being administered, and the Signs are the administration interface, in this case event log messages. Again, none of them are truly static for any significant length of time.

Barron et al.’s terminology clarification attempt further defined 10 features which they considered to be the defining aspects of any information system. These features are given in Table 2.2 along with their natural fit in the representational triangle. The first three root the system in its social context, the next five are to do with the links between Observers/Users, and Real World Objects and Signs. The illustration is nearly complete once the depiction of the *relationship* between RWOs and the Signs is included, with those final *representations* as the instantiation of the knowledge gathered into one or more syntactically representative languages (such as a programming language)[2]. The primary benefit of the representational triangle is its ability to divide the representation of a system from the system itself and depict the different approaches taken by users to those two separate things, as well as providing the ‘big picture’ of the social cause and effect. Barron et al. have therefore provided a framework which includes *context*, i.e. the world in which we live.

The lack of social context or user recognition in event log message systems spurred this search for examples of thought into interfaces. Bear in mind that event log messages consist of textual content and have no graphical element, so interface research into GUIs was not relevant. The iterative processes of refinement that have improved GUIs over the years have not occurred with textual interfaces. This point was made clear by a diagram in Barr's thesis [4] depicting the dichotomy between the designer's mindset and that of the end-user, shown in Figure 2.3 on the next page.

While Barr was looking at metaphorical interface elements in computer games, the diagram shows the same gulf that exists in event log message systems; that of a designer (and/or programmer) with a scope that is entirely separate from that of the user. Metaphors such as icons are used to bridge the gap, yet there is an undeniable possibility of miscommunication. As the representational triangle illustrates, all signs are bedded in social/cultural context, and derive their meaning from it as well as feeding back into the context. An example might be the floppy-disk icon, which has become synonymous with the "save" function of desktop software despite actual floppy disks becoming practically extinct. The floppy disk still lives on as part of the social context - a sign which has meaning separate from its real world object, and gains value simply from its uniqueness [6].

The disappointing conclusion here is: event log messages have not benefitted from any of the considerations presented so far. Those examples in Figure 1.1 on page 23 are brutal evidence of a world-view where the "User" portion of Barr's dichotomy (Figure 2.3) does not even exist. No attempt has been made to embed those signs in a social or organisational context, or even to adapt them for a non-programmer (of that particular program) audience. The language chosen has no standardised interpretation, and comes from an almost infinite set of choices because there is no 'ontology' capable enough.

2.2.2 Ontologies: applying semiotics to, and providing structure for, vocabularies

According to the literature surveyed, the most common application of ontologies appears to be the provision of a set of signs for a specific purpose, all of which are then strictly defined and provided with a degree of context. This approach is evident in Bunch et al. [52], a paper on the monitoring set-up within a rocket-fuel manufacturing plant, where ontologies are provided for notification scenarios and incorporated into their KAoS policy tool. The primary purpose of ITIL [53] (a set vocabulary and standardised best-practice procedures for information technology management) could be said to be the same thing. Matheus et al. use the definition of "a specification of concepts and relationships among the concepts that can exist in a given setting" [9, pg. 547] and inform us that ontologies are well-established in the disciplines of philosophy and linguistics.

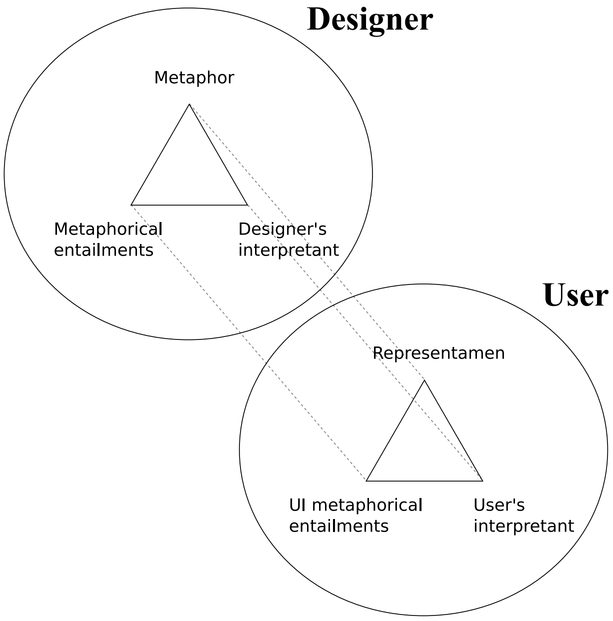


Figure 2.3: The dichotomy between designer and user [4]

While applying an ontological approach to event log messages could be beneficial, it would be premature to do so without even briefly considering that approach's rich background. Stephen Littlejohn, in his book "Theories of human communication" [54], outlines the two main schools of thought as "actional" and "non-actional". The actional school takes a more humanist stand and denies the idea of destiny: people "create meanings, have intentions and make real choices." [54, pg. 29] They recognise that change is inevitable and that there is an element of chaos in life which leads to differing decisions even in (apparently) identical circumstances.

Non-actional adherents, however, hold a pre-destined view [54]. According to this camp, one's DNA and surroundings can deterministically produce a result, i.e. a law could be written that X+Y=Z where 'Z' is a resulting human being which conforms to a template, created from 'X' DNA and brought up in a given situation, 'Y'. This point of view may appeal to rationalists who might then disagree with Derrida's criticism of philosophy on the basis of "logocentrism", "the supposed rational power of the word to explain the world" [6, pg. 88], in other words, the belief that words or signs represent a super-set of reality. An example of non-actional thought might be Doeben-Henisch and Wagner's argument [5] in favour of formalising models of natural language and the representational triangle in order to produce 'provable' models. Their Required Domain Model, shown in Figure 2.4 on the next page, is an example of the emphasis on observation and ennumeration of factors in relation to a human being. It demonstrates a positivist/essentialist mindset in which only directly-observed factors may be taken into account.

Considering again the actional side of the debate; remember that here we espouse that

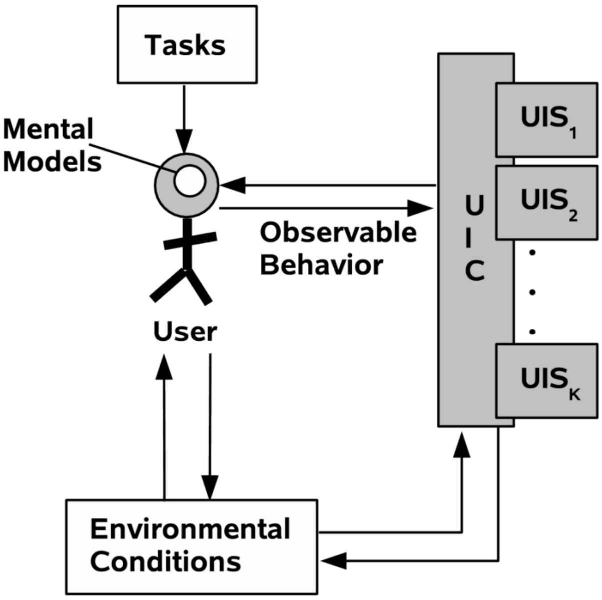


Figure 2.4: The "Required Domain Model" [5]

people bring their own meanings to signs/words rather than those same items being defined on a more absolute scale. Cobley and Jansz reside in this school of thought - they chronicle the thoughts of Jacques Lacan in writing; "the phenomenon of 'différance' encapsulates quite nicely the way in which we delude ourselves into thinking we are rational beings with a firm grip on the process of signification… 'Différance', by its very nature, resists attempts to halt its flow" [6, pg. 98], going on to state that such ideas can be upsetting to those who desire to sit outside the 'social level' (see Figure 2.2 on page 39) and manipulate semiotics as though they were independent of culture and human nature. Umberto Eco is also cited, in particular his 'arctic civilization' example of mis-interpretation, wherein a civilization which survived a future apocalyptic event by living under the ice cap later drew absurd conclusions in their archeological research when pondering the meanings of our artifacts (i.e. our 'signs') [6]. This example demonstrates the extent to which our signs rely on contextual awareness and knowledge.

Any application of ontologies to event log messages would therefore have to deal with the breadth and skill level of the audience, somehow finding a small set of words (remembering that it is a purely textual interface) to act as *unambiguous* signs. The first, and most obvious road-block, is the language barrier: there are systems administrators speaking practically every language on Earth, with a small percentage of them having (currently internationally-dominant) English as their first language. The nuances of English vocabulary are often lost on non-native speakers through no fault of their own - asking them to distinguish between the relative merits of "critical" vs "alert" or "warning" is imposing a hefty burden. Then there is the issue of creating a monotonically-increasing scale from words (such as those just given) and achieving consensus on their meanings, even for native speakers of any language. This is a global problem simply because software is used and standardised across almost all cultures and languages existing today.

Ontologies therefore likely have limited potential in this area (i.e. especially open-source software). Buckley and Siewiorek [25] were disappointed by the poor quality and inconsistencies of logging systems under the control of one organisation (Digital Equipment Corporation), so the likelihood of a word-based scale showing good results is poor when dealing with volunteers from far-flung locations around the world.

2.2.3 Semiotic engineering: tips for creating commonality

Semiotic engineering (SE) is a relative newcomer to the HCI arena and focuses on the end-to-end communication between the program designer and the user. This idea relegates the actual computer interface to a 'designer's deputy'; a proxy which communicates the decisions taken by, and the world view of, the designer [55]. Semiotic engineering therefore regards itself as much more all-encompassing than User-Centred Design and cognitive models, with De Souza and Leitao writing that such established theories "only deal with the user's actions, not the designer's" [55, pg. 3], and likening the learning of a new interface to gaining fluency in a human language, albeit with vastly lessened combinatorial complexity. The root cause of these challenges, they allege, is the lack of consideration given to the designer's goal of communication, i.e. the degree to which the user actually received the message being sent.

The basis of SE is the explicit recognition that "the role of the receiver is as important as that of the sender" [55, pg. 16] - another result of Peirce's representational triangle - because designers/programmers can only communicate via signs and the receiver brings their own interpretation to each sign. This is made even more challenging by the notion (from semiotics, as applied by De Souza and Leitao) that human meanings evolve over time but meanings/signs encoded into a computer program remain static at least as long as that software revision is used, which in some cases may be decades. These static signs are then further limited by the mechanistic methods available for their portrayal. The upside is that static signs can be inspected and evaluated in great depth. But what should they be compared to? Each and every human being represents a moving target when it comes to sign interpretation (and creation of internal meaning) [ibid]. This is the core difficulty in creating a common meaning.

De Souza and Leitao offer a considered retreat from this seeming impossibility. They define 'communicability' as the capacity for the deputy to communicate the essence of the designer's *idea*, perhaps at the cost of detail. The following quote (emphasis De Souza and Leitao's) outlines how the effectiveness of such 'metacommunication' can be observed:

> "... it suffices that one of two things happen (sic) when users interact with computer systems: either that designers mean to tell *something* to users (i.e., to get users to behave in a particular way as a result of being exposed to intentionally produced signs); or that users take a particular course of action because they believe they are *being told* something that justifies their behavior." [55, pg. 17]

In other words, the (meta)communication is effective to the extent that it results in the user receiving a message, or even *thinking* that they are being given a message, but always with an outcome of action (note that this does not have to be exactly the intended action). Taken in this light, the overwhelming majority of event log messages are pointless because (a) the systems administrator either never sees the signs, or (b) does not believe/has great uncertainty about whether they are being given any message, and so takes no action. The barrier of manual interpretation without any reference scale for importance actually hides the designer's/programmer's message. A slightly cynical response might be that those involved in producing the software used by systems administrators actually view themselves as the consumers of event log messages, but such a view would implicitly leave no interface for an administrator - rendering the software a 'black box'.

Another take on the quote above might be: if systems administrators are to "believe they are being told something" [55, pg. 17], then that belief must rest not only on an objective and absolute set of signs (i.e. non-word signs: hopefully free of issues to do with interpretation), but also has to "[justify] their behavior" [ibid] in an organisational and human sense. That is, an objective and absolute scale (for the programmer/designer/sender to indicate their message) must be rendered relative to organisational and human demands (the sysadmin/user/receiver's world view). A number scale is also indicated as a result of the finding that interpretable signs change over time with regard to the human meanings attached to them [55]. This seems to be the best alternative towards the goal of establishing a commonality of meaning.

2.3 Domain-specific limitations

The image appearing here is one of a data set which is subject to semiotic difficulties (from the point of view of filtering tools), most often through a rapid (and usually undocumented) change, but also from differing understandings between the programmer/designer and their audience, systems administrators. The tools available to us for dealing with these issues are inadequate; relying either on absolutely exact matching either for elimination or flagging (i.e. regular expressions) or producing probabilistic results (i.e. data mining, neural networks).

2.3.1 An external examination

The only recent academic critique found of the methods we are using was in the field of bioinformatics; Terri Atwood lamented the unsuitability of exact matching techniques, including regular expressions, fingerprinting, etc, as well as probabilistic methods like Hidden Markov Models (HMM), for the purpose of searching genetic-sequence databases [56]. Like event log messages, a genetic-sequence data set is often enormous and repetitive. Conversely, they possess the advantage of being able to take 'snapshots' that remain static. Atwood's external and frank examination of the tools produced by the computer science establishment unfortunately concluded that none are sufficient for the task at hand.

Atwood identifies "reliability" as lacking in all approaches, while making the case for reliability as a crucial requirement, given the high cost of false negatives and the overwhelming quantity of false-positives the moment that search terms are relaxed. In fact, the issues she faced were remarkably similar to those involved in event log message analysis, especially the way in which the desired search result may very well contain a tiny and otherwise inconsequential corruption/mutation, rendering highly-deterministic "discriminators" useless.

An interesting distinction exists in Atwood's proposal to resolve the situation. Unlike the classical computer science approach, i.e. improve the tool/algorithm while not touching the source data, she advocates the manual annotation of genetic sequence databases and then searching the annotations together with the raw data. In other words, Atwood remains open to algorithmic improvements but would prefer improvements in the source data (via metadata), as painstaking and time-consuming as that may be. While not noting salience *per se*, Atwood's contribution may be in pointing out that we can only locate the most *relevant* records by actually searching information about relevance, instead of wasting time trying to mechanistically infer such an index from raw data.

2.3.2 Lack of state

To further illustrate Atwood's conclusion; other industries have managed much better in their management of events and the construction of a 'nominal state'. Event log messages tend to not indicate or contain state information – in many cases they are received after the event has been completed – which prohibits the construction of a Finite State Machine to represent a system. Even the entire concept of a 'nominal state' is absent. Nominal state refers to the ability to portray a system as either nominal or in fault to some degree, most often via a traffic-light metaphor.

Bunch et al. detailed a NASA research program implementing the KAos Reactive Monitoring and Event Notification (KARMEN) from 2002 onwards, at a hydrogen rocket fuel production plant [52]. Using a subscription model, KARMEN incorporates organisational policies and rules about groups of alarms (i.e. individual alarms occurring together may indicate an overall state) into a structure using independent software agents. In comparison with RFC 5424, only four levels of severity are used: "Critical", "Warning", "Advisory", and "Log". In addition, the well-understood nature of the industrial plant allowed Bunch et al. to construct thorough and hierarchical ontologies of known failure modes and their prerequisites.

There are, however, three main glaring differences between the NASA fuel plant scenario and that of Unix event log messages: (1) the fuel production plant is a relatively static collection of machinery which operates under largely the same physical principles and designs as used for many years, and (2) all inputs to the KARMEN monitoring system are *quantified*, numeric scale measurements. Every input has a lower bound, a nominal range, and an upper bound. The plant operators do not have to decide whether or not "Request is a replay" is an important message. Finally, (3) the inputs reflect a physical state read at a particular point in time.

This level of quantified monitoring permits sections of plant, or even the entire complex, to produce a 'traffic light' status indicator which can communicate an overall situation at a glance. Of course, we must bear in mind that an actual system such as a rocket fuel plant largely possesses complexities that are physical in nature. They are therefore more amenable to human intuition in spotting and fixing bugs long before the entire complex is brought on-line, although deep domain knowledge and experience are often required. Unlike software code, the range of outcomes is considerably more limited, the domain is not one of abstraction, and therefore a 'lurking timebomb' is less likely.

The use of monitoring systems for industrial plants is not a new idea. Efforts date back at least as far as the 1960s; W. E. Willison produced a comprehensive architecture for power-generating stations in 1963 [57], for example. Willison's article specifies many analogue mechanisms, as digital interfaces were only just appearing on the market at the time, but many of the hazards we face today were identified, such as rare failures making it difficult for technicians to retain necessary knowledge and the potential cognitive overload associated with too many data sources. Willison outlined the ways in which a nominal state could be bounded within maxima and minima and methods for aggregating alarms - techniques available in 1963 for industrial monitoring but still not possible for the vast majority of event log messages in 2011. It is clear that a quantification effort, and even more importantly, improved instrumentation, are the only ways to achieve manageability. Note that "improved" should not be read as "more".

2.3.3 Human factors

Willison's fears about the tendency of automation to result in overwhelming amounts of data being collected have certainly played out in other industries. Molloy and Parasuraman's work on vigilance in aviation [58] identified the same issue. Pilot workload has only increased over the years as more instruments have been added to aircraft cockpits, with human intuition being sidelined in favour of automated data collection, the monitoring of which is a task we humans are not optimised for [ibid]. Specifically, the problem revolves around the repetitive observation of hundreds or thousands of variables, which can completely saturate a human's cognitive abilities when a failure occurs (even when alarms are automatically issued as the parameters breach nominal limits).

The very rarity of failures actually makes the necessary level of vigilance much more difficult to achieve [58, 57]. The combinatorial complexity of the data collection/monitoring system itself means the data cannot be assumed as correct, leading to second-guessing of the output, i.e. false-positives due to a failure in the monitoring system designed to alert operators to failures. In relation to systems administrators, the outcome is one of disregard for the data, as it often represents either no correlation or only a weak correlation with reality. A prime example of this can be seen in Pinheiro et al.'s study of Self-Monitoring, Analysis, and Reporting Technology (SMART) data [59] which was harvested from the hard disk drives (reported as "more than one hundred thousand" [59, pg. 3]) in Google's data centres. SMART is an industry standard intended to improve diagnostic communication between the hardware (i.e. hard disk drive) and the operating system software, for the purpose of early detection or even prediction of hard drive failures. In this case, the failure mode is not usually one of false-positives, but false-negatives: SMART data overall provides practically no useful/reliable warning of impending disk failure [ibid].

While it is true that a select few SMART parameters *do* correlate highly with 60-day failure rates, around 36% of disk drives fail catastrophically with no warning whatsoever [59]. It is easy to see, therefore, how sysadmins in charge of over 100,000 disk drives would come to disregard SMART data - in effect it becomes a distraction. This is not a case for eliminating such data collection, but it is one for allowing humans to utilise their intuition for *weighting* the data fields. It also makes a good illustration for the difference between "active" and "latent" errors.

In his influential book "Human Error" [60], James Reason lays out the distinction. "Active" errors are high-profile, obvious failures or mistakes [ibid]. They tend to be localised and individual in nature - often epitomising the military refrain "everything has to go perfectly right for something to go perfectly wrong". Their 'solutions' are most frequently attempts to remove one or several of those factors that contributed to the 'perfectly right' situation. Indeed, Reason mentions that after-the-fact accident inquiries tend to focus on the active side of the equation.

"Latent" errors, on the other hand, can lie dormant, unnoticed for decades [60]. The most frequent cause of these errors are assumptions and mistakes by designers, those in positions of power, or maintenance personnel [ibid]. Latent errors tend not to have immediate effects but their consequences can be much more severe - active errors often rely on latent ones, i.e. without the latent, we would have fewer active. Under-design, cost-cutting, over-optimising; these are some of the names assigned to latent errors if they are un-earthed. The Y2K issue was a classic latent error. The desire to optimise and save individual bits led, decades later, to billions being spent in an effort to update both software and hardware.

The difference between the two categories can be most easily seen by comparing their circumstances. Active errors are most frequently errors of judgement in situations of great pressure and distraction, i.e. we are often unsure whether we could have made better choices under the circumstances. Latent errors, meanwhile, mostly involve a concious, rationalised decision often in full knowledge of the potential consequences. If a server hard disk crashes and this results in data loss, there are many possible errors to consider. Active: a systems administrator not noticing the pre-failure messages in an event log message report, or perhaps not acting fast enough to restore a degraded array of (redundant) disks so that redundancy was restored. Latent: a sysadmin not doing/testing backups, an equipment vendor cutting corners on voltage-smoothing componentry, or perhaps most likely; the disk vendor performing inadequate testing, relaxing quality standards, or not sufficiently funding the research and programming that is required for an effective set of SMART algorithms and disk firmware.

Reason has several theories that relate to such issues. He writes that "much of the work of human-factors specialists has been directed at improving the immediate human-system interface (i.e. the control room or cockpit)." [60, pg. 173] While not diminishing the importance of this work, Reason believes it is "aimed primarily at reducing the [visible] 'active failure' tip of the causal iceberg." [60, pg. 173-174] Experience has shown that

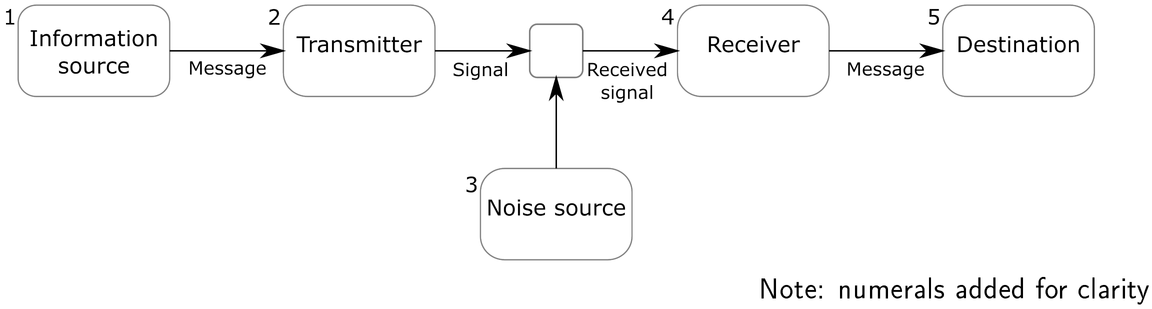


Figure 2.5: Shannon & Weaver's communications model [6, 7, 8]

those 'closest to the coal-face' are not the primary threat to any system - indeed, they are the ones most likely to bear the consequences of any obvious, active errors - but that the risk a person represents rises as their degree of separation to an implemented system rises [60]. The greater this degree of separation, the more obscured and unaccountable their decisions are/were. To put it less ominously, a person far up the causal chain from an implemented system simply has a greatly curtailed ability to clearly communicate their concerns and reservations with those much more closely involved.

2.4 Communications theory

Further to Reason's rationale on causal chains, the communications process itself can be analysed for its role in creating errors. Claude Shannon conceived the communications model [7] in Figure 2.5 to deal with the challenges of radio communication in the late 1940s: a time of entirely-analogue communications technology. Warren Weaver subsequently used it in the context of semiotics and human communication [8], helping to formalise the notion that information was encoded in a lossy manner every time it was communicated [6].

Shannon's and Weaver's fundamental proposition was that the message being sent was rarely (if ever) the message being received. Weaver in particular applied this logic to all human communication regardless of technology; a semiotics problem engendered by language and its shortcomings, but also images or any other form of imperfect communication. The following item explanations are drawn from Weaver [8].

1. **Information source**: The source has a finite number of possible alternative messages from which to choose, due to the limits of any established vocabulary or symbol recognition. Once chosen, this message is sent via the transmitter.

2. **Transmitter**: The transmitter encodes the message into a signal so that it is then capable of crossing a medium.

3. **Noise source**: Noise encapsulates all the unintended alterations made to the signal before it reaches the receiver.

4. **Receiver**: An "inverse transmitter" which decodes the signal and produces a message (note: this is not necessarily the same message as that transmitted).

5. **Destination**: A parallel to the information source which should be able to make sense of the message.

As Weaver put it: "[w]hen I talk to you, my brain is the information source, yours the destination; my vocal system is the transmitter, and your ear and the associated eighth nerve is the receiver." [8, pg. 4] Shannon's and Weaver's communications model neatly portrays the ambiguity that inevitably arises from the use of a finite set of messages, such as a word vocabulary, for communication.

2.5 Summary

This chapter reviewed the most relevant literature around communication via event log messages. Salience was outlined and definitions from other fields of research examined. Methods for establishing common meanings have been considered (including their communication) and the limitations peculiar to the event logging environment presented. It is clear that the salience of events and information has received considerable attention but not in the computer science discipline – rather, we have inherited the legacy of ad-hoc standards which now govern us by virtue of their well-established code bases, the presumptions of which are not being challenged by academia. The Glasgow Coma Scale and "stakeholder salience" are but two examples of revolutionary thought that can lead to real improvements in how we deal with constant flows information of which some is relatively more important.

In order to cope with the lack of obviously- and directly-applicable literature, it is necessary at this time to construct a basis for further reasoning. The problem must be clearly defined. That definition has to be embedded in an appropriate context. From there, a foundation can be established using the scaffolding of literature surrounding the issue: is there a promising way to improve the salience of event log messages that result from an automated filtering process? This will be the focus of the next chapter.

Chapter 3

Resorting to first principles

How can we directly and deterministically improve the salience of event log messages that are the result of an *automated* filtering process? That is, exactly what is it that systems administrators, i.e. the audience for any deployed system event log, actually consider to be important? The previous chapter informed us that other disciplines have established a notion of salience, that multiple dimensions can be better than one, and that it seems that any solution should allow the observer's own context to be applied when determining salience.

The salience of the content emerging from previous efforts has been subjected to only the most simplistic of tests (if any), such as a small number of 'domain experts' that offer their opinions on how "good" or "useful" the output is to them. For example, Saniefar et al. utilised two unidentified people to determine the percentage of terms that were "really relevant" [42, pg. 775]; whereas Wallin et al. were advised by only a single network operator, stating that "[the operator] indicated that priority estimates that were within one step of the true value would be useful" [10, pg. 19]. It barely needs to be mentioned that these reference sources do not meet any known scientific standard in terms of the sample taken, the miscellaneous factors influencing the decisions of the domain experts (which should be controlled for), or repeatability – their advice embodies the term 'unproven presumptions' [51].

In one final case, the rationale for a crucial decision was left entirely unexplained by Yamanishi & Maruyama: "[h]ence in the evaluation of this experiment we formally define a failure symptom as any alarm that is raised within one week before 'lock-up'. This definition *seems reasonable* from the standpoint of network operation" [18, pg. 504] (emphasis mine), illustrating a (stated) presumption that alarms are no longer salient after a period of one week. They also relied on a single human operator for manual assignment of salience to events, to wit: "the 'lock up' was the failure which a network operator taking care of the network systems thought most critical and was mainly concerned with" [ibid]. To be fair, however, there is no available research on the salience of event log messages, leading to the use of these unscientific sources.

Flack & Atallah [40] came at the problem from a different angle: their paper dealt in part with the preservation of information from audit logs, as they were parsed into canonical forms using various approaches. Flack & Atallah were concerned that the parsing process was discarding a degree of the semantic value contained in log messages - making future analysis that much more difficult and error-prone [ibid]. They identified the fact that messages contain meaning which is embedded in their grammatical form alone, and that it can be imperilled even by the first step of any solution that requires parsing (i.e. anomaly detectors), leading to a situation where such solutions are working off of a 'lossy' input. When a message is parsed, it is usually split up into constituent fields and such a separation of data points represents a loss of information; much like the pieces of a jigsaw puzzle versus the completed whole.

Flack & Atallah proceeded to iteratively develop a grammar-based parser for Sun Microsystem's BSM audit log format to the point where it could interpret almost every possible message [40]. Note that an audit log is a strictly-defined log format that only records explicit user actions transiting through the system kernel - such as deleting a file or listing a directory of files. The set of possible actions is comparatively small, thus audit logs are not as complex as the free-form natural language content of event log messages. Essentially, their efforts revolved around catering for all identifiable special cases and resolving ambiguity wherever possible. One problem, however, with such an approach is that it rapidly approaches or even exceeds the maintenance burden of a policy/signature-based filter, in terms of coping with the inevitable system or software changes. This is even before the anomaly detection stage, and its associated algorithmic challenges, is considered. The result: salient messages require yet more time investment to identify.

3.1 So, what is a 'salient' message?

What can be deduced is this: researchers have always had to check with *real people* about whether a given event log message is salient or not, since there is no mathematical model for determining message salience. It therefore follows that the researchers did not believe their own level of domain expertise or experience to be sufficient. After all, the consultation of one or two people does not constitute evidence *per se*. A Flack & Atallah-like exception illustrates that mechanised interpretation principles for log messages cannot be generalised and the parser has to comprise an expert system, effectively creating a policy engine for responding to every possible outcome (which is barely feasible even in a strictly-defined, single-source domain like audit logs). The X.733 standard [43] attempted such an approach for event log messages and was obsolete even before it was released. A state of continual change does not permit static handling.

To follow a simplistic behaviouralist model for the time being, as well as considering De Souza and Leitao [55], information considered salient by humans is information that either initiates, or influences the path of, *action*. It is clear from the prior examination of semiotics that this salient information will change because it is composed of signs (words) which are dynamic by nature. To simultaneously narrow down the problem and give it relevance, context must be introduced.

The context of this investigation is the role of a systems administrator working with open-source software. Often a professional, the sysadmin is responsible for maintaining, and where-possible, improving, business continuity and capability (this aspect has not changed in decades). Any failure of a critical system will result in urgent work being undertaken and a likely investigation into any active errors committed [60]. In addition, the proliferation and pervasiveness of modern computing is in stark contrast to the 1970s era of perhaps a single-digit number of computers in each organisation (and this is the environment in which event log messages were first conceived), resulting in many simultaneous pressures on an administrator, thereby decreasing vigilance against failures [58]. Sysadmins therefore sometimes find themselves in 'survival mode' - spending all their time maintaining the status quo, with no resources for improvement of the situation.

Informed by context, it is now possible to state that systems administrators consider information to be salient if it requires or influences their action with regard to their responsibilities in the organisation. Salient information would therefore be anything that impacts on business continuity or capability; it takes the form of simultaneous pressures and can unfortunately lead to 'survival mode' if it overwhelms an individual's capabilities. Information that is not salient does not have these impacts, exerts pressure only in the sense that its salience has to first be identified, and should not push one into 'survival mode'. It is necessary to bear in mind, though, that each organisation and each individual sysadmin have differing standards and requirements for the services they rely on and these may change over time (some use the term "intrinsically non-stationary", e.g. [18, pg. 499] to describe the event-logging environment). Any salience *threshold* is therefore a moving target and more can perhaps be gained by using the salient/non-salient distinctions as end-points on a sliding scale, in agreement with Buckley's premises in [26].

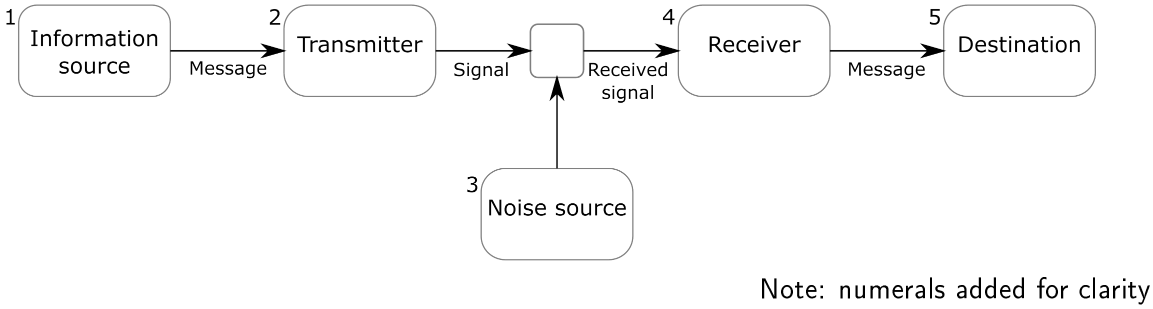
3.2 Modelling the end-to-end lifecycle of an event log message

To set the scene for the lifecycle of a message, it is important to consider some statistical likelihoods. An exaggerated form of the Pareto principle has previously been located in statistical analyses of event data [10]; simply put, a very few distinct events happen frequently, while the bulk (more than 95%) occur rarely or even only a handful of times over several years. Following a 'low-hanging fruit' approach, the nature of humans is to filter the most common events first, and when the most straightforward solution is a simplistic and deterministic regular expression, the elimination of known events (whether known-good or known-bad) is the path of least resistance. This leaves behind a pool of unknown events which belong to either or both of these categories: 'unknown due to rarity or first sighting' and 'common but unknown due to insufficient system knowledge or familiarity'. Such a quandary exists independent of any current filtering technology.

Digging deeper, the issue with the automated filtering stage (in whichever form it may take) between humans and the event log message-generating systems comes from the very nature of the solutions implemented. Humans are poorly adapted to deal with the flood of messages that even one busy system can produce, so the normal coping strategy is to reduce the cognitive load by selectively excluding most or all of the surrounding contextual information on offer, simply to avoid overload. Yet there is one contextual area currently being entirely neglected: the software code itself.

Event log messages are issued from code, of course, but they then have to rely on lossy and ambiguous natural language to communicate as the 'designer's deputy' (see page 44). Code represents the designer/programmer's most direct instantiation of their ideas [61]; it contains (among other things) comments, conditional testing structures, mathematical algorithms and loops. As a formalised representation it is already a limited subset of the thoughts inside the head of the designer/programmer. A million-line program may then cut this back even further, only ever issuing a few thousand different log messages: for example, OpenVPN version 2.1.4 [13] contains 79,710 lines of code, of which 1,311 are for issuing non-debugging event log messages, making for a ratio of roughly 1:61. The cycle of restriction continues with every step, continuously losing context and paring back the value of whatever semiotic signs eventually make it to the other end of the communication process: the mind of a systems administrator.

A model for such fundamentally lossy communication has not previously been applied to a unidirectional interface such as event log messages. Shannon and Weaver's communications model is presented here once again for the benefit of the reader. The model helps



Note: numerals added for clarity

Figure 3.1: Shannon & Weaver's communications model [6, 7, 8]

to visualise Weaver's notion that information was encoded in a lossy manner every time it was communicated [6]. Here I will attempt to further adapt the model - this time into the semiotic context of event logging (see [62]).

1. The **information source** is defined as the original intention or thoughts of the programmer and/or program designer; those individuals editing source code that later becomes the binary executable, libraries, scripts, etc running on systems. The programmer is the person best informed about the situations leading to an event log message being issued – their mental model of the problem domain is the most detailed and in-depth of the actors in this context – and originally creates the sequence of conditionals leading to such an issuing. Semiotics informs us that this level of conciousness cannot ever be communicated without loss because no sign is entirely adequate (the 'picture worth a thousand words' premise). The thoughts of the programmer are often most candidly portrayed in code comments, such as "should never get to here!" or "probably going to fail, but we'll try to recover anyway, for what it's worth". The "message" represents the transition stage of writing code.

   Limitations: human competence, memory, decision-making, clarity, rationality, intuition, reasoning.

   Information state: context-rich, vague, informal human thought processes.

   Example: Hypothetical OpenVPN programmer: "We need to initialise the network tunnel adapters using the operating system - there is no other way. Errors have to be catered for and we have to inform the user if there is an error. This all has to be done in a cross-platform manner with as few platform-specific clauses as possible."

2. The **transmitter** is defined as the encoded thought process of the programmer, i.e. the source code which is then (most often) compiled into binary machine code. Such information is a subset of the original intention of the 'information source', having

been formalised and revised into a Turing-complete grammar, following absolute rules of logic and with conditionals that are machine-testable. Thus the human thought processes have transitioned from poorly-defined but context-rich, to well-defined, testable and context-poor. The "transmitter" is therefore the implemented mechanism which 'transmits' the event log message when all the (machine-testable) conditions for doing so have been met. The "signal" is an individual log message sent by the binary program in question.

Limitations: grammar expressiveness, human language ability (i.e. a native speaker of the applicable natural language, the individual's level of education).

Information state: formalised, testable grammar, probably in a binary form.

Example: shows three message-issuing lines (40, 45, 48) in all of the platform-specific tunnel initialisation code (partial listing in Listing 3.1 on page 60). OpenVPN itself only issues messages in the Win32 platform section, or a generic 'giving up' message if the platform is unsupported. For all other platforms it relies on piping any errors from the `ifconfig` command it runs, which is supposed to configure the VPN tunnel interface in a command-line shell, back to the user (these lines are omitted because they do not have any content of their own). It is easy to perceive that a user might be bewildered by such an error message, presented as-is without reasons or interpretation - as OpenVPN is not creating the message itself and the semiotic context is therefore different. When `ifconfig` is run, it of course does not change its output to match the style of OpenVPN's and most users would likely not be aware that `ifconfig` is being utilised at all.

Another point to note in Listing 3.1 is the nested conditionals consisting of a "switch-case", an "if" statement and numerous variations of "#if". These conditionals create their own context which is omitted from the issued message, perhaps wisely, but omitted nonetheless. The message on line 45 could be suffixed with the following: "`This message was issued because the Win32 API returned a null value in response to your '--ip-win32 netsh' parameter.`" Similarly, the message on line 48 could offer more information on the various platforms that it tried to match already, i.e. explain which conditionals it tested, that then led to issuing the message.

It is true that specific debugging messages can help with these situations, but the use of debugging techniques requires deliberate and focussed action to be taken (i.e. enabling the debug messages, which may be a drawn-out operation on a change-controlled system that is already deployed in production) which can only happen once that action has been identified as a necessary step. For effective *communication* to take place, the error message would have to explicitly stipulate one or all of the following: the reading of the source code, the enabling of debug messages, or telling the user that the message is not important and can be discarded. As a designer's deputy, the code must highlight salient messages that it issues, and when a message is salient it must facilitate the user's understanding of *why* - a line of reasoning which needs to draw on programmatic context as background for the situation. Otherwise the signs being used are semiotically-impotent because no 'real-world objects' (see Figure 2.2 on page 39) are being offered - only isolated and cryptic snippets, analogous to hearing only a single sentence from a long discussion.

3. The **noise source** is defined as both the sources of other disparate messages and unrelated messages from our own binary, as well as any packet loss related to the commonplace use of the User Datagram Protocol (UDP) with logging daemons. Thankfully, Shannon & Weaver's concept of 'engineering noise' has largely been nullified in our domain due to the layering of digital communications, the Transmission Control Protocol (TCP) and packet checksums. TCP is being standardised for event-logging purposes by the Internet Engineering Task Force (IETF) as demand grows for a reliable network transport, despite TCP itself not being ideally suited to the job [63]. The use of UDP as a transport for messages has its advocates too, and their justifications are valid: lossy or congested networks can prevent session-based protocols from working [64, 63], simplicity can be invaluable in emergencies, and UDP retransmission can be tailored to each individual situation [64]. Malicious attacks such as man-in-the-middle (MitM), intentional spurious distractions or merely simple mistakes with firewall rules also cannot be entirely ruled out with any transport and lead to imperfect information (or none at all) being received: these have to be considered as noise contributing to what Weaver calls "undesirable uncertainty" [8].

   Complex multi-tier systems, clusters of machines issuing identical messages, denial-

Listing 3.1 tun.c selective snippet from OpenVPN version 2.1.4 [13]

```
/*
 * Platform specific tun initializations
 */
void
init_tun_post (struct tuntap *tt,
        ...
{
  tt->options = *options;
#ifdef WIN32
        ...
#endif
}

/* execute the ifconfig command through the shell */
void
do_ifconfig (struct tuntap *tt,
        ...

#if defined(TARGET_LINUX)
#ifdef CONFIG_FEATURE_IPROUTE
        ...
#else
        ...
#endif /*CONFIG_FEATURE_IPROUTE*/
#elif defined(TARGET_SOLARIS)
      ...
#elif defined(TARGET_OPENBSD)
      ...
#elif defined(TARGET_NETBSD)
      ...
#elif defined(TARGET_DARWIN)
      ...
#elif defined(TARGET_FREEBSD)||defined(TARGET_DRAGONFLY)
      ...
#elif defined (WIN32)
        ...
        switch (tt->options.ip_win32_type)
          {
          case IPW32_SET_MANUAL:
            msg (M_INFO, "********␣NOTE:␣␣Please␣manually␣set␣the␣IP/netmask␣of␣'%s'␣to␣%
                s/%s␣(if␣it␣is␣not␣already␣set)",
                 actual, ifconfig_local, print_in_addr_t (tt->adapter_netmask, 0, &gc));
            break;
          case IPW32_SET_NETSH:
            if (!strcmp (actual, "NULL"))
              msg (M_FATAL, "Error:␣When␣using␣--ip-win32␣netsh,␣if␣you␣have␣more␣than␣
                  one␣TAP-Win32␣adapter,␣you␣must␣also␣specify␣--dev-node");
            ...
#else
      msg (M_FATAL, "Sorry,␣but␣I␣don't␣know␣how␣to␣do␣'ifconfig'␣commands␣on␣this␣
          operating␣system.␣␣You␣should␣ifconfig␣your␣TUN/TAP␣device␣manually␣or␣use␣an␣
          --up␣script.");
#endif
...
}
```

of-service attacks, continual change and poor documentation: all of these aspects make it more difficult for an administrator to locate a problem signature, and to isolate it from the continuous flow of information; a task not dissimilar to siphoning off one particular drop whilst drinking from a fire hose. Such problems are exacerbated by undesirable uncertainty - the perpetual requirement for a systems administrator to second-guess the validity of data. Weaver envisaged yet another noise source: "semantic noise", contributing "perturbations or distortions of meaning which are not intended by the source but which inescapably affect the destination" [8, pg. 15]. This is at least partially catered for above, in the "message" transition between thought-processes and code.

The 'received signal' is the combination of all these possibilities flowing into a centralised log collector, i.e. a logging server. The hundreds-of-megabytes figures mentioned on page 21 are measured at this point. A factor that must be remembered is that the systems administrator may not have any control over the devices sending event log messages to their central collection server.

Limitations imposed: incomplete information which has possibly been deliberately manipulated or omitted, spurious information (also possibly injected as a distraction tactic).

Information state of the 'received signal': possibly-salient content submerged in a flood of distractions.

Example: a 9-second snapshot on a lightly-loaded system shows an OpenVPN session re-initalisation, shown in Listing 3.2 on the next page. The example is quite 'kind' in that the messages are contiguous in this case. On a highly-loaded system in a large production environment, it is unlikely that they would be contiguous simply because of the volume being received. There is also no relationship between the OpenVPN messages and those surrounding them, but such a data point should not form the basis of any assumption that this would always be the case. Root causes can produce errors further up 'the stack', for example, disk read errors on a database server causing a client on another machine to issue event log messages about database timeouts.

4. The **receiver** is defined as the system processes or daemons which both receive messages, and then perform filtering on them. The filtering process may appear out of place, but is similar in its intention to Weaver's "semantic receiver" which

Listing 3.2 Some OpenVPN messages embedded in a log message flow

```
...
Dec 16 17:35:16 host named[3898]: automatic empty zone: A.E.F.IP6.ARPA
Dec 16 17:35:16 host named[3898]: automatic empty zone: B.E.F.IP6.ARPA
Dec 16 17:35:16 host named[3898]: command channel listening on 127.0.0.1#953
Dec 16 17:35:16 host named[3898]: zone 0.in-addr.arpa/IN: loaded serial 1
Dec 16 17:35:16 host named[3898]: zone 127.in-addr.arpa/IN: loaded serial 1
Dec 16 17:35:17 host named[3898]: zone 16.172.in-addr.arpa/IN: loaded serial 2010042601
Dec 16 17:35:17 host named[3898]: zone 255.in-addr.arpa/IN: loaded serial 1
Dec 16 17:35:17 host named[3898]: zone localhost/IN: loaded serial 2
Dec 16 17:35:17 host named[3898]: zone example.com/IN: loaded serial 2010071901
Dec 16 17:35:17 host named[3898]: running
Dec 16 17:35:17 host ovpn-server[3912]: OpenVPN 2.1_rc11 i486-pc-linux-gnu [SSL] [LZO2] [EPOLL] [PKCS11] built on Sep 18 2008
Dec 16 17:35:18 host ovpn-server[3912]: /usr/bin/openssl-vulnkey -q -b 2048 -m <modulus omitted>
Dec 16 17:35:20 host ovpn-server[3912]: Control Channel Authentication: using ’tls-auth.key’ as a OpenVPN static key file
Dec 16 17:35:20 host ovpn-server[3912]: WARNING: normally if you use --mssfix and/or --fragment, you should also set --tun-mtu 1500 (currently it is 1442)
Dec 16 17:35:20 host ovpn-server[3912]: TUN/TAP device tun0 opened
Dec 16 17:35:20 host ovpn-server[3912]: /sbin/ifconfig tun0 172.16.201.1 pointopoint 172.16.201.2 mtu 1442
Dec 16 17:35:20 host ovpn-server[3921]: GID set to nobody
Dec 16 17:35:20 host ovpn-server[3921]: UID set to nobody
Dec 16 17:35:20 host ovpn-server[3921]: UDPv4 link local (bound): [undef]:1194
Dec 16 17:35:20 host ovpn-server[3921]: UDPv4 link remote: [undef]
Dec 16 17:35:20 host ovpn-server[3921]: Initialization Sequence Completed
Dec 16 17:35:25 host sensord: sensord started
Dec 16 17:35:25 host sensord: Chip: acpitz-virtual-0
Dec 16 17:35:25 host sensord: Adapter: Virtual device
Dec 16 17:35:25 host sensord:   temp1: 32.0 C
Dec 16 17:35:25 host sensord: Chip: vt8231-isa-6000
Dec 16 17:35:25 host sensord: Adapter: ISA adapter
Dec 16 17:35:25 host sensord:   +3.3V: +3.27 V (min = +3.13 V, max = +3.45 V)
Dec 16 17:35:25 host sensord:   +2.5V: +2.47 V (min = +2.37 V, max = +2.62 V)
Dec 16 17:35:25 host sensord:   VCore: +2.08 V (min = +1.89 V, max = +2.39 V)
Dec 16 17:35:25 host sensord:   +5V: +4.96 V (min = +4.71 V, max = +5.22 V)
...
```

he imagined sitting between the receiver and the destination, the difference being that the semantic receiver was to cater to the varying characteristics of each potential receiver device [8]. Receiving Syslog daemons, however, are relatively simple automata which currently do little more than appending the contents of network packets to an appropriate log file.

Here it is useful to step back for a moment and reconsider the state of the information in the flow which has been received. As Weaver specified, the word information as used here is distinct from meaning, in that the very notion of meaning has disappeared by this stage and must be reconstructed from scratch from mere information (akin to data). "Information [represents] freedom of choice and hence uncertainty as to what choice has been made" [8, pg. 11] - which tells us that *more information* can mean *more uncertainty* but this can still be *desirable*. Consider the example of audio CD sampling rates: the virtue of the 16-bit wave format is excellent descriptiveness in that there are so many values with which to describe an analogue audio wave. Compared to a 4-bit wave format, 16-bit is clearly superior. Yet 16-bit creates much more uncertainty in that the probability of any given value being received is greatly reduced.

The automata handling the flow, though, cannot determine a meaningful message from nonsense, with the result that messages like those in Figure 1.1 on page 23 are (by default) piped into a few broadly-specified files, dependent on their Syslog facility value. The common Linux/open-source files would be `auth.log`, `messages`, `syslog`, `daemon.log`, `mail.log`, and so on. On Windows operating systems the three logging categories are Security, System and Application. From the point of view of automata, the only reliable method for determining where a message should be delivered is for the programmer/designer to pre-decide this via the facility value. RFC 5424 standardises these but does not broach the topic of reliably determining which messages a human should see.

The task of reverse-engineering salience into the messages is bequeathed to the filtering process, and this task has thus far been performed with the (valid) assumption that since a message's meaning cannot yet be determined by a machine, we have to make do with matching regular expressions (e.g. Listing 1.2 on page 25), or using statistical methods – usually boiling down to trained data mining. This is a crucial weakness of the natural-language nature of log messages. Furthermore, their unpredictable yet often rapid rate of change means most solutions are obsolete even before coding begins. Whereas message routing was accepted as a function of metadata set into source code, the human-interface side was apparently never considered beyond the very locally-focussed severity levels. We might as well turn our tools to bear on a newspaper's website and rely on them to tell us the 'important' headlines, along with succinct and relevant quotes from the articles, for all the real-world success witnessed to date in the log analysis field.

Limitations: static or trained filtering tools that must assess salience based on predefined rules or trends, probabilistic factors in neural networks, and/or signatures.

Information state: out-of-context messages, littered with potential false-positives and -negatives, and most importantly, devoid of any organisationally-useful or machine-readable indicator of salience.

Example: A centralised log-collection server running software which is almost inevitably using regular expressions (or a comparable variant of static string-matching filters) such as those in Listing 1.2 on page 25. The software detailed in Section 3.3.1 on page 67 is one example. This point of the flow is therefore purely automatic and handled by a finite state machine. Challenging aspects like encryption, authentication, session management and others outside the scope of this thesis are dealt with here.

5. Predictably, the **destination** is defined to be the systems administrator. This role is filled by a human who must make critical decisions on a daily basis, often using the information presented by the various automated monitoring processes. Such a role requires vigilance – a difficult and error-prone task when there might be only one failure indicator amid the logs, and the administrator has other business tasks to complete [58]. There can be no guarantee of the administrator being a specific domain expert, let alone being experienced with any particular (sub-)system - an administrator 'covering' for another who is on vacation is but one counter-example.

Limitations: busy human with other priorities competing for their attention, experience level and language ability of said administrator. Uncertainty about the method of information presentation (E-mail? Website? How often is it checked? How much time or opportunity is there for a human to verify the information?), i.e. interface assumptions that may not be suitable for the end-user.

Information state: mental impressions formed by the information of questionable salience that has made it through the filtering process.

Example: any systems administrator who may or may not take action as a result of a message reported to them. In this OpenVPN working example, the sysadmin has to decide the importance of the message "`WARNING: normally if you use --mssfix and/or --fragment, you should also set --tun-mtu 1500 (currently it is 1442)`" and proceed from there. The real-world system this message was taken from has been running an OpenVPN server daemon for over six years, during which the warning message has been displayed around twice a week (on average). No problems have ever been noticed as a result of the warning.

Specifically, the message relates to *where* IP packets will be fragmented: either 1500-byte data packets will be cut up to fit into valid VPN packets which then travel over a layer-2 media with a maximum packet size of 1500 bytes, or the data packets are packed into >1500 byte VPN packets which themselves then have to be fragmented to fit on the media. In the implementation examined, the former alternative was chosen, but OpenVPN issues the warning message despite no problems being evident. A design (i.e. Syslog) requiring this level of detailed consideration for every possible log message, including ones that have yet to be issued, is not feasible in today's environments, and even less so in tomorrow's.

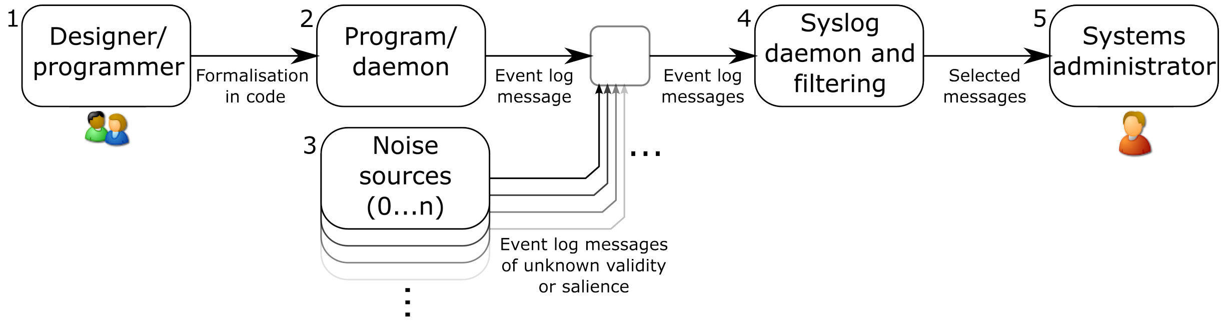


Figure 3.2: Adapted version of Shannon & Weaver's communications model; for event log messaging

It is clear from the process above, depicted in an adapted form in Figure 3.2, that a communication deficit develops between the source and the destination. The argument presented here is that this deficit is a factor in dulling the vigilance of administrators; creating a risk of "latent failures" [60]. Merely learning the nominal state of systems becomes an exercise in futility, as can be seen in the many 'dashboards' that seek to visually portray a current state and yet fail utterly to do so [14] – and through such a systemic lack of awareness, failures can result [58].

3.3 Mechanistically interpreting natural-language messages

A common thread among the academic literature is the difficulty involved in interpreting natural-language portions of event log messages. Not only is there the immense challenge of parsing a language itself, but the complications arising from non-standard formats, non-native users of any given language, in consistencies across and within products (e.g. [25]), and even spelling mistakes, are far from trivial. The general approach to anomaly detection can thus be summed up as "rare == bad" (which can be observed in [65, 17, 18]); an assumption that results from the abandonment of the actual interpretation of meaning.

Such an approach is not reflected in other well-established disciplines that were detailed earlier (industrial chemical plant monitoring failure [52], failure monitoring in aviation [58], and electricity-generation equipment monitoring [57]) that tend to focus on nominal states, rates of change and threshold trips. Those examples rely on quantified measurements wherein the scales themselves depict salient points. That is, the scales have meaning in and of themselves. Without quantified *and* useful scales available to us, the approaches taken to event log messages universally rely on identifying 'bad' messages; in isolation to begin with and occasionally seeking correlations with other messages at a later point in time.

It is true that policy/signature-based tools can complement the results of frequent-pattern data mining, which left to its own devices would consider a weekly time-synchronisation issue to be more serious than one hundred disk-write failures every day, but the results of such regular-expression filters still fundamentally reflect the competence of the filter set and not an objective, or even subjective, evaluation of importance. The statistical or neural-network approaches feature only probabilistic outcomes and a need for training which, even when drawing from knowledge bases, seems unable to surpass a 50%-correct threshold (as in [10]).

These issues resonate with Atwood's findings [56], and those of a large-scale literature survey (of more than 150 published papers) by Facca and Lanzi which concluded that the most promising approach for useful Web usage data mining was an end-to-end conceptual schema that improved the quality of the source data [66]. Filters which use a statistical or neural-network technique also tend to be limited to academic settings. In summary, effective filtering is still a 'hard' problem [67].

3.3.1 Examining a common deployment: Logcheck

To know where one is going, one first has to know where one has come from. For a working example, an unmodified installation of Logcheck v1.2.69 [20], the most-often installed policy/signature-based alternative [21] under Debian Linux v5.0 [68] was examined. In this context, "white-listing" refers to matching for the purpose of discarding a message as unnecessary/known-good/routine, while "black-listing" is the opposite practice of matching known-bad messages explicitly for notification purposes.

The 'server' monitoring profile contains 1324 lines of regular expressions for white-listing innocuous event log messages, 47 lines termed "cracking" that black-list suspicious messages, 12 lines for "violations" black-listing (often used for flagging emergency conditions such as disk failure) and 155 "violation-ignore" lines that white-list specific cases otherwise flagged as violations. These numbers are from the rules shipped with the Logcheck package - they do not include any rules added by other software packages. The Logcheck software uses this repository of regular expressions to filter event log messages, which by default it does daily, resulting in an e-mail to the administrator. This e-mail contains all the messages which either matched a black-list expression or did not match a white-list one.

It should be noted here that there are several issues with this simplistic approach: there is a presumption that *all* unanticipated 'bad' messages will pass through the filtering process; that *all* high-priority messages will be flagged by the "violations" or "cracking" black-lists; and that the white-list will be maintained so as to reduce 'noise' in the daily report and therefore call attention only to genuine issues. Noise generally consists of the non-salient and/or duplicated messages created by node 3 in Figure 3.2 on page 65.

Rules such as the small sample in Listing 1.2 on page 25 are typical of the mechanism used for policy/signature-based tools. These particular rules are representative of the enumerated whitelist that removes messages which administrators do not need to be made aware of – every line represents a special case deemed to be acceptable. The blacklist rules are written identically and are only identified as 'bad' by the file they are stored in.

The ratios of whitelist-to-blacklist rules are indicative of the potential for regular expressions to cope with unanticipated situations. Logcheck's 'server' profile possesses a total of 1479 message whitelist rules that eliminate known-good messages, compared to only 59 blacklist rules that highlight known-bad messages. This ratio of roughly 25:1 clearly illustrates the usefulness of the regular expression mechanism: when it comes to system failures, it is only possible to anticipate highly-specific cases, and indeed Logcheck (as of v1.2.69 in Debian v5.0) only does so with disk monitoring errors and a very few cases of kernel events. An anonymized sample of real events which didn't match any of the default

Logcheck filter sets is shown in Listing 1.1 on page 23. These lines were e-mailed to the system administrator as possibly-serious events.

Notably, compared to the 'server' profile characteristic, the 'paranoid' monitoring profile caters for increased administrator paranoia by removing whitelist elimination rules, without adding blacklist rules. Choosing this profile results in a daily e-mail with an increased number of spurious events. As another point of comparison, the LogWatch [19] package in Debian Linux v5.0 contains a default filter set of 1104 regular expressions, the vast majority of which are used to count event 'hits' in log files and whether a given hit is considered harmless or not is particular to each log file input. In only 39 cases is a "bad" counter incremented. This represents a whitelist:blacklist ratio of roughly 27:1, as of LogWatch v7.3.6.cvs20080702-2.

3.3.2 Problems with the current approach

A fundamental issue with the policy/signature-based situation as it stands, is the emphasis on removing known-good event log messages, flagging or counting known-bad messages, and the implicit assumption that an acceptable outcome is for any remaining events to fall into a category of 'unknown'. The number of unknowns can be iteratively reduced with the addition of regular expressions to the filter set, but this is a considerable maintenance burden in modern environments [10] and any automated attempt to generate these expressions has a significant and unavoidable risk in regard to false positives/negatives [56].

Human involvement cannot be avoided altogether [56, 65]: although an algorithm can identify trends in data, it does not and cannot know exactly what a human observer is interested in. Worse yet, this effort is being repeated in parallel in every organisation which utilises log analysis tools, as the final burden of interpretation and filtering is on the end-user (i.e. systems administrators). To a limited degree, however, the filtering effort always has to be customised according to the needs, policies and priorities of each organisation, and thus this capability should not be removed, but instead the need for its use minimised.

Whichever approach is used, it can be difficult to categorically justify the addition of such unknown events to a white- or blacklist, leading to a situation where the distraction 'noise' level in a filtered event log incrementally builds over time as the software base changes. An example of this dilemma is a message from the Berkeley Internet Name Daemon (BIND) [69]: "`clients-per-query decreased to [a two-digit number]`", for which this author has not been able to locate a conclusive result regarding severity or priority, despite many searches.

Such a known vs unknown conflict is not helped when the data input source represents a continuous flow that is continuously changing (chaotic in nature), incomplete, inconsistent [25] and semantically heterogeneous [70] – thereby defying any attempt to reliably/deterministically correlate events, such as [15] but especially ontological efforts such as [71, 70] (see Section 2.2.2 on page 41 for more details of ontologies). Event correlation relies on a relatively static and well-understood environment; for the purposes of identifying root causes [72], and optimally, downstream impacts. Indeed, Yamanishi & Maruyama use the term "intrinsically non-stationary" [18] to describe the modern situations within which event logging systems exist.

This 'moving target' becomes even more relevant when considering research in the healthcare sector which asserts that interoperability between disparate systems cannot take place in a meaningful manner unless valid relationships are identified between the systems' ontologies [73], simply because ontologies that are continually in flux cannot maintain those relationships in stasis (by definition). Current automated techniques to cope with this reconciliation challenge are far from complete [30] and event correlation (in the sense of automatically linking events across and between systems) remains a 'hard' problem [18].

Other attempts at automatically and independently assigning priorities to events have not met any greater success: the correlations between system-assigned problem severity and actual real-world problem priority appear to be weak at best, and practically non-existent when examining event type versus real-world priority [10]. Such a disparity is partly due to a given device or system daemon issuing event log messages with no knowledge of, or regard to, its own place in an organisation's infrastructure. In short, a piece of computer code is written with only an awareness of the component's functionality, and not with any prescient foresight as to how salient its messages will be to the administrators receiving them. This information, an indexed indicator of salience (as opposed to severity), is absent in every event log message yet seen.

As somewhat-anecdotally identified by Wallin et al. in [10], systems administrators utilise their domain expert knowledge of the object in question (this author would add that this includes a historical perspective of system issues) and its infrastructural importance, as well as the time and type of the event, when evaluating importance/relevance/priority on a per-event basis. This is of course contingent on being able to reduce the volume of messages to a reasonable level [63]. And yet, as identified earlier, there are messages which even experienced administrators are bewildered by due to the pace of change in the computing sector and the depth of knowledge required.

With their neural network and trouble-ticket database combination (i.e. attempting to make use of organisational knowledge derived from tickets), Wallin et al. found a "statistically significant improvement for operators compared with the currently available alarm severity" [10, pg. 19]. By their own admission though, the outcomes were probabilistic rather than deterministic, and the system could provide no justification whatsoever for *why* a particular alarm had been highlighted. The neural network remained a 'black box'. Another point of failure is the trouble-ticket database: Wallin et al. make optimistic assumptions regarding the validity of the information here, and as covered earlier, determining the true salience of an alarm often requires knowledge and expertise which may not be available within any given organisation. As such, the solution posited by Wallen et al. attempts to 'make do' with information that is, from a semiotics point of view, incomplete and likely misconceived. It does not attempt to improve information quality, convey meaning from the program developer, or otherwise support human decision making.

A natural upper bound therefore exists on the usefulness of expert systems, neural networks, customised policy/signature-based tools, data-mining and machine-learning, when the role of the verifier or trainer is taken by people who themselves cannot categorically determine the desired outcome in all cases. Fundamentally this comes back to the competence of those who choose or clean the training data set – and this is a weakness of many academic studies, e.g. [15, 10, 16, 17, 18], which proceed with a *static*, vetted data set, rather than one in a constant state of random flux. It is all very well to optimise an algorithm (in the broadest sense) for messages that are two years old, but quite another to engineer one that can adapt to new and unforeseen situations without a specialised team of researchers on-hand.

Log messages cannot be classified with the same simplistic, binary strategy as e-mail spam. Regarding the problem as information *triage* is more valid. The current solutions do not recognise and cannot cope with the constraining factor of limited human attention, so it follows that they are inadequate at triaging event log messages. Would any reader of this thesis be genuinely comfortable with the scenario of being wheeled into a hospital emergency department and having their Glasgow Coma Scale assigned by a computer running an expert system based on regular expressions, or a data mining algorithm with only "probablistic" results?

3.4 Quantifying information saturation

To quantify the problem at hand, and due to the apparent lack of any prior such work, the raw information content of event log messages from a Los Alamos National Laboratory (LANL) Cray XT supercomputer [74] were analyzed using Weaver's information theorem, shown in Algorithm 3.1. This is to demonstrate the scale of information that the current

Algorithm 3.1 Weaver's information theorem [8]

$$H = -[p_1 log p_1 + p_2 log p_2 + ... + p_n log p_n]$$

solutions are attempting to handle. The 'syslog' file (named messages.sdb), which receives all kernel and daemon messages, was used for the analysis.

It is important to remember that Weaver defined 'information' in this context as "freedom of choice". The set of all possible values that may be received is indicative of the depth of information; when this set is larger, a set of data can contain more information. Binary is thus the practical minimum as far as 'information' goes. Trinary or quaternary (e.g. DNA) codes *can* contain more information in the same number of bits. The more possible values there are, the more uncertainty there is about which value was sent from sender to receiver - this can be considered 'good' because the greater freedom of choice enables a more descriptive communication about an analogue world. Noise also increases uncertainty but simply in a 'bad' sense; the result (an increase in uncertainty) can be identical but from a different cause, yet the receiver cannot distinguish between causes [8].

The natural counter is redundancy - and indeed Weaver states the redundancy of the English language as around 50%, which enables us to communicate even when words are lost or misheard. That redundancy in natural language is an advantage for a human listener with a 'wet-ware' brain to effortlessly process it. The same redundancy is a liability for algorithmic computing, however, as algorithms cannot discard it and instead flag even tiny, semantically-trivial alterations as differences just as significant as any other. From a semiotics point of view (see Figure 2.2 on page 39) algorithms process signs as, and only as, data; they have no genuine experience as observers, nor any genuine knowledge of real-world objects, i.e. context. After all, signs are the only interface for algorithms. Human-level language interpretation is still a problem reserved for strong AI.

3.4.1 Saturation results

Since it is the information of the messages themselves that concerns us rather than any artificial uniqueness, the first five columns of event-log data (time-stamp and machine name) from LANL's Cray XT [74] log files were removed. All 621494 messages in the event log file were considered, with the most frequent occurring 9804 times. Using each message's probability of occurrence (p1...pn) produced a value for H of *16.126* (3 d.p.). The maximum attainable value of H for 621494 records (i.e. each record having equal probability) is *18.552* (3 d.p.). This gives a ratio of roughly *0.869* (3 d.p.), meaning

Data source[a]	Number of messages	Ratio of H
0809181018	3,397,749	0.851
0810082020	51,951	0.967
0810151644	97,644	0.906
0811011951	878,503	0.830
'Typical' Munin monitoring tool	18,931,584	0.512
'Typical' Debian v5.0 e-mail servers	41,080	0.922
Default Ubuntu 10.04 desktop	24,947	0.926

Table 3.1: Other ratios obtained with Weaver's theorem

[a] Numbered sources are available from [74]

that the variability/freedom-of-choice in the event log file was 86.9% of the theoretical maximum.

To verify the results of the initial experiment, subsequent runs were performed, some with data from sources other than LANL's Cray XT. The additional results are shown in Table 3.1. Note that input data was only obtained from systems running a Linux kernel; this was to reduce skew from the different kernel logging frequency one might encounter with a proprietary UNIX or open-source BSD kernel. Time-stamps and machine names were consistently pruned at the first stage.

The one application-specific log file examined was from an open-source statistical monitoring tool named Munin [75], which graphs metrics such as hard disk temperatures. As expected, the variation in messages was greatly reduced in such a specific domain, which consists entirely of highly-repetitive messages concerning graph generation and authentication to hosts for the purposes of statistics collection. The 'typical' Debian v5.0 setup was a two-machine cluster running Postfix, Cyrus IMAP and Spamassassin. The Debian and Ubuntu systems only retained their `/var/log/syslog` files for seven days, as per default settings, so all the available contents were analyzed.

While the results show some variation amongst similar systems (i.e. Linux kernels with a largely-to-entirely GNU userland), and a much more significant gap to the more uniform sample (Munin), we can be confident that the generally high ratios show a highly-packed information space. In other words, they indicate that event log messages represent a remarkably rich source of 'information' (by Weaver's definition of information representing freedom of choice), due to a high uncertainty of which message may arrive next.

The uncertainty referred to here is 'desirable' in that it results from the vast array of choices available to the sender [8], yet it makes the filtering task a hard problem. Simply put, the burden of automated log filtering could be greatly eased by reducing H, meaning the destruction of information, by cutting the number of choices available to any program issuing log messages. Strangling the event log output like this could only be achieved by permitting a certain (restricted) selection of words and phrases to be used - a technique which culminates in immediate or even pre-empted obsolesence, as in the case of X.733 [43].

Systems are only becoming more complex and permanently 'strangling' them would inevitably cut information vital to those with sufficient domain expertise, when troubleshooting is necessary. The author and Internet Engineering Task Force (IETF) contributor Marshall T. Rose put it best when he stated that troubleshooting is a 'fire-fighting' operation and that what is appreciated during those times is an unobscured simplicity[64]. The aim here is *not* to obscure information or reduce communication via a restricted vocabulary, but rather to enable salient information to be automatically recognised.

3.4.2 Implications

The situation systems administrators find themselves in is that the sender's information represents a large proportion of non-incidental noise; the noise source is largely the sender itself, or groups of other senders (see Figure 3.2 on page 65). The underlying-yet-unstated assumption that the receiver wants all the information transmitted by the sender, though, is flawed in this case. Weaver suggested the concept of "semantic noise" in part to deal with such issues – "the perturbations or distortions of meaning which are not intended by the source but which inescapably affect the destination" [8, pg. 15]: a concept which finds echoes in Buckley & Siewiorek's observation that event log messages represent information fused with assumptions [25].

The underlying subtext in such messages is the assumption of equivalent knowledge on the part of the consumer (here, the systems administrator) compared to the programmer or program designer. No metric, other than the coarse and ambiguously-worded Syslog severity scale (Table 2.1 on page 32), exists to alleviate the burden of such source-code-level knowledge acquisition. Many would opine that "critical" is worse than "alert", as might "error" be, but the Syslog scale disagrees. Wallin et al. wrote that the system-determined severity correlates poorly with priorities in organisations, so much so that it is largely disregarded [10]. This is not to say that the scale is without its uses, however.

The Syslog scale is indeed the closest that systems have come to quantifying their log output, but it limits output to an entirely component-centric view and does not deal with several important aspects: how the component fits into the organization's infrastructure (i.e. important vs superfluous); the impact an event could have on that infrastructure; or the certainty of the programmer/program designer about the situation surrounding the issuing of the log message. These issues imply that we need to add metadata (as Atwood [56] advised for genetic sequence databases) so that events that systems administrators consider to be important, i.e. salient messages, can be more easily distinguished from noise.

Moreover, we have been attempting to filter data based on the data itself. This is analogous to categorising news stories into "current events", "sports", "lifestyle", and so on, by searching for signature phrases or using a data-mining algorithm. Much of the event-log-message job is little more than a matter of opinion, e.g. "bad", "don't care", "normal", "crisis" but this is perhaps the most difficult task of all for a machine filter. It is simply easier and vastly more accurate with such a data-set for a human to provide categorisation metadata, via tags or the like. The filtering of event log message data is not generally a problem involving potential adversaries, as it is in web censorship (e.g. [67]) or spam filtering, since the programs running on one's systems are not assumed to be malicious and in the event of a system being compromised, no output (i.e. data *or* metadata) could be trusted anyway.

If the metadata can be trusted, then the task of interpreting the natural-language portion of the message can be largely discarded. Such a highly-packed and redundant information space could be left for those best suited to deal with it: humans; while machine-readable metadata is used to trivially whittle down and tune the volume received. The reader may view this as a 'retreat' from the problem of language analysis, and it is, but such a retreat is a necessary step given the explosion of operational systems and their log output over the past 15-20 years, a period during which minimal advances have been made in textual analysis. The advent of strong AI would likely permit this ground to be re-taken - after all, what else could acceptably handle Weaver's concept of semantic noise?

3.5 Summary

This chapter has attempted to illustrate the event log message problem from first principles. The lack of relevant literature stipulates a more adventurous approach and significant reasoning to reach the point where established literature can again be drawn on. A working definition of salience has therefore been presented in the context of a message lifecycle, followed by a real-world example of message filtering, and finally the immense density of information that must be dealt with. This leaves the question: where do we go from here?

Chapter 4

Seeking an organisational context

> "How should I know if it works? That's what beta testers are for. I only coded it."
>
> – Attributed to Linus Torvalds, somewhere in a posting
> - *fortune* package from Debian Linux v5.0

Wallin et al. [10] described system-assigned 'severity' as correlating poorly with 'priority' in a larger context. But what is 'priority'? For their part, Wallin et al. only offered one anecdote of the network provider they studied: "the event time, managed object, and alarm type attributes in combination with [the individual's] own experience and lookups in support systems" [10, pg. 8]. The most directly relevant theories actually come from the business disciplines, specifically management.

Stakeholder salience (first introduced on page 36) is the notion of assigning priorities to competing demands for one's time when dealing with external influences. Mitchell et al. defined a three-dimensional scale to recognise and quantify the pressures which influence decisions made about the claims of stakeholders. 'Power' is the ability of a stakeholder to cause a change in their favour, 'legitimacy' measures the wider perception of a stakeholder's actions, from the point of view of established societal and cultural norms, and 'urgency' communicates the genuine need of the stakeholder for their claim to be looked at. In this chapter, the stakeholder salience concept will be applied to the comparable factors weighed up by systems administrators - although a critical divide does exist in the sense of *agency.* Agency is used here with the sense of autonomy, in that an agent can perform actions according to beliefs and for its own benefit [76, 77]. Logging systems are simplistic message routers and lack any notion of *agency* (by this definition).

Stakeholder dimension	Characteristics of stakeholders (entities *with* ‘*agency*’)	Characteristics of message salience (messages being lines of free-form text *without* ‘*agency*’)	Currently available in message metadata?
Power	● Scope: **individuals/representatives** ● Ability to bring about change ● Can be self-perceived	● Scope: **individual component/daemon** ● Measure of how noteworthy an event is ● Self-perceived within programmatic scope, i.e. the scale is calibrated to each program's extremes	Yes, as “severity”
Legitimacy	● Scope: **society, organisation** ● Not a self-perception ● Based on norms, values, beliefs	● Scope: **organisation, business** ● Enforced by external factors, i.e. manager ● Based on policies, norms, service-level-agreements (SLAs), etc	No
Urgency	● Scope: **each decision point** ● Multi-faceted; time sensitivity vs criticality ● Concerned with *risk* and potential exposure	● Scope: **each conditional statement** ● Multi-faceted: validity/confidence in assumptions vs anticipation of situations ● Concerned with quality of input data to each conditional; i.e. probability of a valid/correct test in all circumstances	No

Table 4.1: Relation between the measures of stakeholder salience [1] and event log message salience

4.1 Power → severity

The first dimension in Table 4.1, 'power', has a straightforward parallel in the existing severity scale. Mostly obviously, the scope is similar in that power is vested in individuals and severity is assessed within the context of an individual daemon. Each of these has its own context, or 'ecosystem', of factors, assumptions and priorities. Just as 'power' can be perceived by and in a person - and therefore calibrated by that person's experience - 'severity' is calibrated according the extremes encountered in a given program. Buckley commented on this pecularity of scales; they tend to be defined by their endpoints, with 'normalcy' residing at the mid-point of the range [26].

4.2 Legitimacy→ impact

To discover a parallel for 'legitimacy' the parameters must first be defined; here we are searching for the characteristics of salience when it comes to the real-world impact or relevance of an event log message. Not the message itself; but rather the event it was issued to describe and communicate. The event has some level of impact on the organisation(s) within its range. Mitchell et al. quote Suchman for their working definition of legitimacy: "a generalized perception or assumption that the actions of an entity are desirable, proper, or appropriate within some socially-constructed system of norms, values, beliefs and definitions" [1, pg. 866].

If we subsititute an automated system for the entity in question, we can say that a 'legitimate' system is one that acts in a desirable manner, calibrated by a socially-constructed and established set of norms, etc. But the system/entity itself is not the intended target of this application; the *event notifications* the system generates are to be measured individually against a calibrated scale. The system can be classified in typologies such as [2] and judged in a larger social context as set out in [78] given that it is a (relatively) static entity with the resulting distinct attributes. Event log messages merely give witness to events that the system's programmers/designers thought notable for any number of reasons, and as single lines of free-form textual content, do not have distinct attributes beyond character counts and character set/encoding.

It therefore follows that the primary parameter in the social/organisational context is the *perceived impact* of events which result from the formalised programming of the automated system. Despite the immediate semantic disconnect, this concept is remarkably similar to the "perceived severity" [43] of X.733, in which it is the perception of those interacting with the system that matters. These ideas revolve around the real-world impact of events, i.e. the cumulative outcomes, rather than the severity of each particular data point. A disk fault may be a particularly severe error simply because a physical piece of equipment needs to be replaced, but when it occurs in a redundant disk array (i.e. RAID) specifically set-up and configured to cope with that problem, user perception of the fault is likely to be entirely absent. Redundant RAID features exist to maximise system availability by allowing faulty disks to be dealt with in a timely manner, with no interruption to service and therefore no *impact*. Essentially these arrangements allow for *faults* without flow-on errors or failures. The fault is catered for just as error-correcting-codes exist to cope with inevitable errors in data transmission or storage.

'Impact' is not a measure of potential consequences should a machine or service fail. Mitchell et al.'s 'legitimacy' doesn't work in such a way either. Their scale is for assessing the cumulative standing of an external claim on managers' time; 'impact' quantifies the cumulative estimation of an event's effects *as described*. The process for determining an event message's impact value therefore excludes speculation – the generation of false positives must be avoided – and includes these factors:

1. The known/expected (not speculative) functionality consequences for the entire software and/or hardware stack to which the event's component belongs. The 'component' is the item of software or hardware to which the event has happened.

2. The (non-)existence of redundancy or comparable measures to cope with errors or failures. This requires ennumerated configuration knowledge to be available to the decision tree issuing messages. Effectively this factor encompasses the capability to avoid a decrease in service levels which could be perceived by a system user.

3. Knowledge of organisational priorities for the event's component. Such priorities, by definition, will be different for each organisation. This is perhaps the most important factor, because it is the strongest break with severity; it widens the scope of existing logging techniques to include a human norm/value system; and allows complex relationships between components to be expressed via the assignment of similar prioritisation levels.

One factor explicitly not included is the likelihood of the estimated impact. To combine the estimate and the certainty of that estimate would be overloading the scale, resembling a statement such as "the flood will have a likely peak and variable range within three metres" rather than "the flood will likely peak at three metres, plus or minus half a metre", the latter being rather more useful.

4.3 Urgency→ certainty

So far the concepts of severity and impact have been divorced through recognising their differing scopes and contextual demands. As amply demonstrated by the quote at the beginning of this chapter, though, programmers and designers often entertain doubt due to external factors beyond their control or knowledge of the assumptions embedded within their code.

The notion of certainty in this work was originally inspired by literature on, perhaps surprisingly, ontologies for enhancing military battlefield awareness. An early distinction to bear in mind is the nature of the problem when dealing with battlefield events: a "situation" is bookended by event notices with their own timestamps, requiring a system which regards events as entities: long-lived and possessing their own attributes [9]. Event log messages, on the other hand, tend to be instantaneous notifications of events which are usually completed within nanoseconds, so the notification is only written to disk (at best) tens of milliseconds later, i.e. an eternity in computing time.

Matheus, Kokar and Baclawski introduced their "core ontology for situational awareness" [9] with the goal of improving situational awareness through making salient information easier to identify. Their conceptual result is shown in Figure 4.1 on the following page. The "certainty" attribute of "PropertyValue" is one they justified as so:

> "In real-world situations sensory information is not always accurate. To account for this there needs to be a way to represent the certainty/uncertainty inherent in sensory data; this becomes particularly important if the system using the data intends to perform data fusion or higher-order reasoning" [9, pg. 549]

Matheus et al. apply their concept of certainty to individual attributes; meaning that they assign varying certainties to values from different sources. The underlying purpose is to portray the unavoidable margin of error that accompanies quantification rather than discarding such information. Margins of error, of course, surround any variable which purports to exactly represents an analogue or subjective source. These margins are naturally expanded when a variable is used outside its original purpose, as can happen when re-purposing a data source; for example, the infamous Imperial vs metric units conversion issue which led to the loss of the Mars Climate Orbiter spacecraft. Data intended for one purpose was re-used without an adequate verification of its properties (in the case of MCO, the unit). Such a verification would have increased the *certainty* of receiving *valid* data from that source.

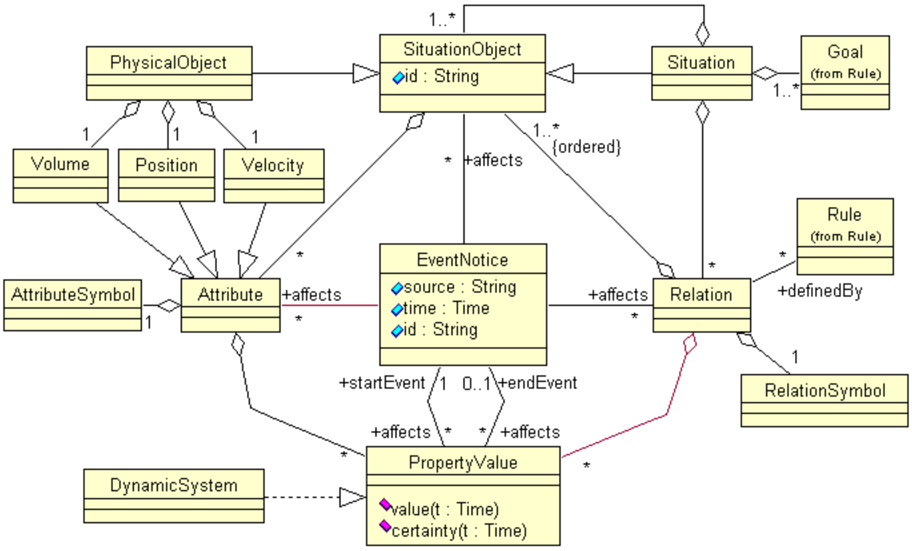


Figure 4.1: Core SAW Ontology [9, pg. 547]

4.3.1 Certainty as validity, or confidence in assumptions

The ‘certainty’ measure appeals because it can convey information that has never before been codified by the programmer/designer. Every conditional structure is built on the assumption that the variables being tested are appropriate for their purpose; such as in industrial plant design, where measurements like pressures and temperatures can be delivered correctly by the sensor but unintended outcomes can still result if other unanticipated factors contribute to a problem. The Three-Mile Island nuclear accident, for example, was in large part caused by an operator assumption that the reactor vessel relief valve would close when power was cut to its solenoid (an “active” error [60]). In fact there was no sensor to detect whether the valve was closed or not - the manufacturer had also assumed the valve would close in the absence of solenoid activation, and *also* assumed that the operators would be sufficiently experienced to incorporate temperature and pressure readings from the pressure-relief piping that the valve controlled, into their determination of correct operation (a “latent” error [ibid]).

Retaining an indicator of certainty in such cases would retain the influence of *doubt*, a factor which some consider to be invaluable, as explored in [79]. Airline pilots are another case, often being required to corroborate instrument readings with information from other sources [58] as an explicit admission that doubt exists in instrument readings. Indeed, programmers and designers incorporate such techniques as far as they are able; this is sometimes betrayed by their informal comments in source code. The Linux kernel is a fine example - many device drivers must cater for tremendous levels of doubt when initialising or reconfiguring hardware which contains foibles/bugs or simply doesn’t follow established standards. One of the greatest throwbacks in PC architecture is the Basic Input/Output System (BIOS): it is a well-known source of surprise issues, compounded by non-standard implementations of such standards as Advanced Configuration and Power Interface (ACPI), themselves comprised of assumptions built on older assumptions. Anyone who has experienced hibernate or suspend problems (i.e. notebook/laptop power management) with an open-source operating system has seen the results of compounded assumptions first-hand. To act as though doubt does not exist, or to proclaim that it can be ignored, is to invite future error.

At a more basic level, consider that every situation-dependent event log message is, by definition, issued as the result of conditional tests. An `if` statement comparing the variables `RealTimeClockRate` (i.e. RTC) and `HighPrecisionClockRate` (i.e. HPET) for equality may then result in the issuing of a success (or failure) message. But how valid is the test to begin with? Any pre-determined tolerance threshold carries with it personal evaluations that should be expressed. Not only that, but the sampled measurements may not sufficiently cater for fluctuations in the instantaneous clock rate, and if they do, how certain can we be of what is “sufficient”? Entertaining doubt like this entails the acknowledgement of risk. Ignoring doubt is tantamount to taking the risk of painting over rusted metal: sooner or later, reality catches up with us. The ‘certainty’ measure attempts to convey this contextual information to the end-user (i.e. systems administrator).

These everyday decisions made by program creators may seem insignificant, but their event log messages still clutter up log files on innumerable systems. The “higher-order reasoning” Matheus et al. write of is exactly what today’s log filtering systems attempt to do, though without any knowledge of certainty. Particularly concerning is the requirement for program code to issue messages (for later filtering) in situations which have not yet occurred. This is the second hurdle for the concept of certainty in logging systems.

4.3.2 . . . as anticipation of the future

The second aspect of certainty is not concerned with the immediate validity of data and assumptions, but rather the applicability of the reasoning itself, regarding the troublesome tendency of ‘real life’ to alter situations and make them inconceivable to past anticipation. The recent Fukushima nuclear power plant disaster in Japan is a fine example of this “if only we had done X” quandary. To an extent this uncertainty can never be ameliorated because it involves a degree of crystal-ball gazing. Many applications, though, are written in environments where the designer/programmer is aware that the ‘rules of the game’ may very well change.

The question the program writer needs to consider is this: “how certain am I that the situation this test is predicated on will always hold?” An author of a anti-virus program, or intrusion-detection system (IDS), may choose to automatically categorise all detected attacks. Some examples might be “SYN packet flood”, “port knocking attempt”, “buffer-overflow attempt”, “Trojan horse”, “privilege escalation”, or the like, all of which are well-known techniques. There is of course no natural law which prevents an entirely new form of attack being devised - one which then could not be automatically categorised as the appropriate conditionals couldn’t be envisaged ahead of time.

A conditional structure in this sense (e.g. a nested switch-case) is limited not only by the input variables being tested, as discussed above in section 4.3.1, but by the very code it is composed of. Anti-virus ‘engines’ are frequently updated to deal with exactly this problem. Anti-virus signatures, on the other hand, are analogous to the input variables. If we accept source code as a static snapshot of an individual’s or team’s impressions at the time, it is easy to perceive that such a snapshot will inevitably drift from reality as reality moves on; a problem also faced by practitioners of semiotics, especially those whose attempts are aimed at freezing an absolute set of definitions [6].

4.3.3 ... as a parallel of ‘urgency’

Urgency as presented in Mitchell et al. is a human imperative. It relies on *agency* as the reason for its existence; urgency is only found in situations where humans are depending on a specific outcome. Given that event log messages and the static automata that generate them lack their own agency, what remains in its absence?

To strip away context, dynamism, knowledge, opinion, and all other characteristics of sentience and culture, leaves us only one third of the semiotics representational triangle (see Figure 2.2 on page 39). More precisely, the signs (i.e. syntactics) are all that remain. Barron’s ten features of information systems, shown in Table 2.2 on page 40, offers us an equivalent term: “representation”. This is the level of perception of the computer; encodings as representations, signs without meaning, action without awareness.

The scope of Mitchell et al.’s ‘urgency’ dimension is always limited to a particular claim or relationship [1], and it additionally appears to be the most temporally-curtailed (i.e. shortest-lived) dimension of the three. Urgency changes minute-to-minute, decision point to decision point, giving it the highest granularity in comparison with ‘power’ and ‘legitimacy’. This point can justify its application to each or any conditional statement in code.

Yet perhaps most importantly, urgency is forward-looking. Power is grown to be exercised, legitimacy accumulates only to be drawn down, while urgency anticipates future success or failure in order to *reduce potential risk*. The reader may wish to pause here and contemplate the semiotic signs used to communicate and represent risk. Common responses might include best and worse cases, margins of error, and statistical likelihoods (e.g. the percentage of people who will contract a medical condition by age 60). All of these are representations of (un-) certainty, making 'certainty' the unifying concept/notion for signs which originate at a highly-granular source code level. So, when communicating the foreseen risks of individual conditional statements that by the very nature of source code are entirely syntactic, an indication of 'risk' would be too overloaded by connotations of individual investment and agency. Certainty succeeds as a clearer delineation between risk's contextual concerns, and the syntactic competence and foresight possible in source code.

4.4 Summary: three dimensions for salience

The sections above have outlined the rationale for adapting Mitchell et al.'s measures of stakeholder salience into measures of event log message salience. The varying scopes and other analogous characteristics were first compared in Table 4.1. How can these dimensions be drawn together, though? A mere re-labelling of Figure 2.1 on page 37 leads to Figure 4.2, illustrating the intersection where the *most salient* messages reside; identical to Mitchell et al. with their stakeholder claims. This does not sufficiently portray the distinctions of scope between the dimensions, though.

Figure 4.3 offers a layered interpretation of the dimensions. 'Impact' is the richest context and exists in the human situation surrounding the systems administrator: it is the larger picture outside the computing system that exerts all the pressures and dictates the priorities which decide how the sysadmin spends their time. Many of these are outside the sysadmin's control, but again, the actions of administering the system are controlled by this context.

Further modifying Weaver's communication theory to illustrate the flow of communication from scope to scope, the 'impact' scope is centred around item five in Figure 4.4, although it possibly envelopes item four as well, since the Syslog aggregation and filtering point only enacts the policies determined by the sysadmin. Typically there will be a few sysadmins in each organisation; they will be aware of each other's abilities, and policies can serve to standardise organisational priorities amongst them.

An important distinction between Figures 4.3 (the "traffic light") and 4.4 (the adapted communications model) is this: the traffic light is hierarchical; it depicts the dimensions

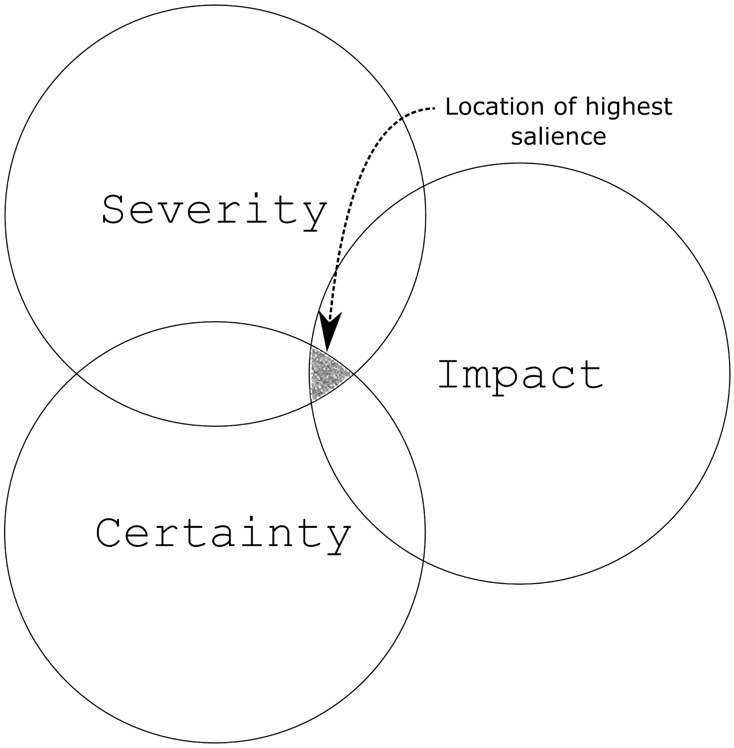


Figure 4.2: Dimensions possessed by salient event log messages (adapted from [1])

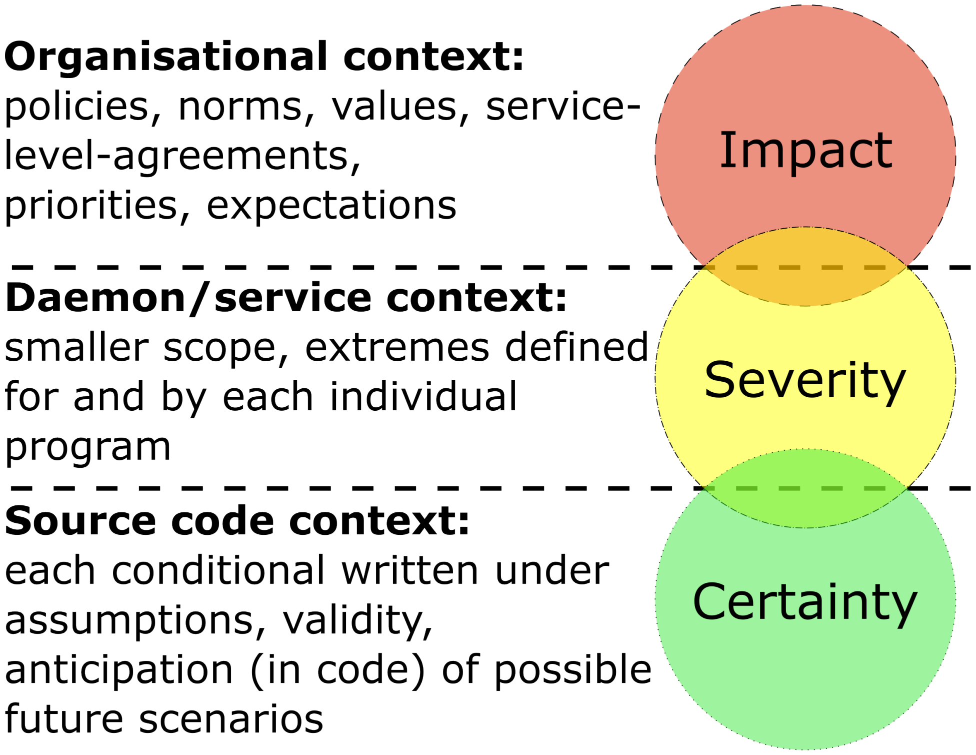

Organisational context:
policies, norms, values, service-level-agreements, priorities, expectations
Daemon/service context:
smaller scope, extremes defined for and by each individual program
Source code context:
each conditional written under assumptions, validity, anticipation (in code) of possible future scenarios
Impact
Severity
Certainty

Figure 4.3: Scope and context of each dimension

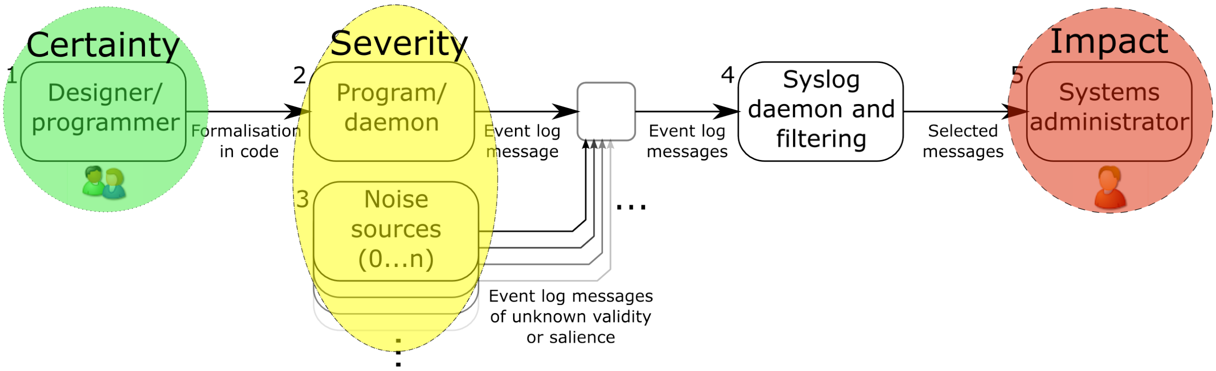


Figure 4.4: Adapted version of Shannon & Weaver's communications model; for event log messaging (with dimensional scopes)

in relation to each other, ordered by relevance to an organisational context. Figure 4.4 depicts the dimensions in terms of their conceptual *origin*, i.e. where the core scope is within the communications flow which includes the source and destination for messages. 'Severity' has its origin in the daemon scope - which is what defines the extremes of the severity scale - and is therefore often different across daemons, as detailed earlier in this document. It is safe to assume that there will be *many* sources of messages in an organisation: between ten and several hundred per server, and at least one per switch, router, printer or other such embedded device. In comparision, the organisation will likely have only one 'impact' context (due to established policies), but it will be sitting atop *many* individual 'severity' contexts which originated in *other organisations*, such as vendors or open-source developer groups.

'Certainty' still remains as the largest conceptual challenge. It has no known precedent in event logging. Its scope is in every individual conditional statement that leads to the issuing of an event log message - rather than being calibrated by the entire daemon/program as whole, in which it resides. With the multitude of programmers and designers involved with software projects (for example: the Linux kernel, as of 2008, had over 1000 developers committing code between every release [80]), there are likely to be *many* individuals per daemon/program, but more importantly, it is also safe to assume that there are *many* conditionals per daemon/program. The designer/programmer writing each statement is best placed to rate them on 'impact' (initial values only) and 'severity'. The 'certainty' value simply reflects the confidence of that person that the values they have assigned will be correct for the future situations where the program is being used.

Chapter 5

Enforcing organisational realities

Faced with a paucity of suitable research related to event logging in organisations, the question had to be asked; is it possible to adapt successful strategies from other research areas? Parallels in other established disciplines were thus sought for any examples of useful methodologies. Specifically; any research which had significantly improved outcomes while dealing with familiar domain limitations. The following characteristics of event log messages were used in this search:

1. A data set without the statistical nature of a 'population', and which therefore does not accurately match a distribution [26];
2. can be intentionally skewed to provide a distraction or reinforce false impressions;
3. often without documentation or substantiating evidence [25];
4. an unpredictable, rapidly changing (perhaps chaotic) stream of information, at least on significant time scales [18];
5. and thus requires constant work to maintain an automated filter[10].

The financial stock market model was determined to be the best fit for these characteristics: the "efficient market hypothesis" certainly makes the case for unpredictability in stock markets (with perhaps with the exception of very brief, i.e. millisecond, time scales) [81] - something that can be observed as "fashionable" market trends take hold and a market bubble develops only to "pop" later, in a cycle which has been recurring since the 1970s [79] (applies to points 2, 3, 4 above). Stock markets are commonly 'attacked' for personal gain; trying to fathom the whys-and-wherefores of the market is a business in itself; and the data does not fit any definition of a natural population but more closely resembles the Pareto principle [10] (points 1 & 2). A further attraction point lies in the ongoing development of automated, autonomous trading agents such as those from Sherstov and Stone [82]: software designed to extract profit without direct human involvement, and the rapid iterative cycle involved in trying to gain a competitive advantage in such a complicated market space (points 4 & 5).

The situation surrounding the stock market is arguably more complex and harder to comprehend than any computing environment, even leading to attempts (such as Zhai et al. [83]) to automate the prioritising of such diverse information as media reports. While there are several parallels to be drawn, such as the widespread industry use of simple 'trading rules' [81] which as signature detectors closely resemble the regular expressions used with log filtering, one must note the dissimilarities too.

Event log data cannot be placed on a ticker or meaningfully graphed (one only has to look at the history of dashboards to see how poorly suited a condensed format is to their presentation of data [14]). There is no quantified mean, nominal state, or 'norm' to tell a sysadmin the state of their computing systems at a glance, as can be done so trivially with stock prices and indices; which effectively communicate state according to a shared (industry) 'norm' or perception of value. Event log systems do not contain actors or autonomous agents that receive feedback and that are presumed to possess motivations or beliefs [76, 77], nor do they engage in competition. Many of these discrepancies make the problem of analyzing event logs less hard, whereas the lack of instrumentation and, therefore, quantified analysis makes it more difficult. Messages tend to be written in natural language, cryptic data field formats, or both. Yet there is no reliable and automated way to, on a daily basis, determine which particular messages a systems administrator would want to see, and therefore filter out the millions of messages deemed unimportant.

Finally, salient log reports are not yet a reality [62], yet the world's financial markets are increasingly governed by the rules in automated trading agents [84, 85], with some estimates placing the trading volume controlled by them as high as 90%. These agents deal with a never-ending and time-critical stream of data; some of the factors behind their success are examined and those factors' applicability to another such stream of data (event log messages) is considered in the following section.

5.1 Stock market parallels

The afore-mentioned 'trading rules' are analogous to the regular expressions most commonly used to filter log messages (e.g. in Logcheck [20]): mechanised rules which apply a simplistic conditional test. Such rules can optionally be clustered together to make use of tuples [82] and incorporated into automated 'agents'. Evolutionary algorithms can be used to then select the best agents across many thousands of generations, as done in [81], or they can be selected through the results of tailored, empirical testing, from distinctly different, handcrafted algorithms, as in [82].

It is at this point that we run into the first of several distinctions between event logs and stock trading; the fitness test (either automated or manual) for the stock agents is very simple indeed: did the agent make a profit over the course of a day [82, 81]? The test for event logs, though, is one of salience [62]; a contextual definition and not a thing which can be instantiated in an `if` statement. Thus, comparisons of 'state' are easily formalised for any trader of stocks ("is the price higher now than before?"), whereas they are practically non-existent for computing systems and their event log messages.

Another parallel that was initially identified, the rapid cycle of iteration, soon turned into a discrepancy upon closer examination. Stock traders alter their autonomous agents with the aim of increasing profit, i.e. taking advantage of any change in the price of a targeted stock, and/or even changing that price through their own actions. A rapid rate of code revision is driven by the desire to maximize profit versus the competition. The agents also receive feedback and use reward mechanisms to reinforce profitable behavior [82]. This is highly distinct from the iteration cycle that systems administrators are accustomed to: they simply have to accept the messages that are issued and cannot effectively alter the contents.

It is either impossible or difficult for a sysadmin to fundamentally alter the components in a running system. For closed-source systems this is a given, in that the end-user has no ability to change the system, and may very well be legally prevented from doing so. For open-source systems the only alternatives are submitting a code patch which, if accepted by upstream developers with differing motivations, represents a global change, or maintenance of a customised code branch: likely to be a heavy burden indeed. The examples in Figure 1.1 on page 23 are strong evidence that users of software are not always able to positively influence the textual content of what are highly technical event log messages. The iterative feedback loop, whether automated as a reward mechanism or involving a human programmer/designer, is the most promising feature of trading agents, given the comparative absence of this feature in event logging systems.

Software is heavily reliant on standards, whether they be 'open' like TCP/IP, or 'de-facto' like the current dominance of the Microsoft Windows® ecosystem on desktop computers. The systems administrator stands to gain little from non-standard software - interoperability is diametrically-opposed to competition. Stock-trading systems, on the other hand, benefit from unique and proprietary approaches in a competitive environment. It is difficult to imagine a community of Wall Street stock-traders all co-operating on an open-source trading algorithm which they then all hoped to benefit from. Nonetheless, the model of rapid iteration appears an encouraging method for independently honing and improving an information flow, which in many other respects resembles that of event log messages.

5.2 Introducing rapid iteration to event logging

The current situation, illustrated in Figure 5.1, involves a third party who usually resides outside the systems administrator's organisation. This third party is the designer/programmer (i.e. developer) responsible for the software product being used by the sysadmin. As the 'gateway' for any improvements or refinements to the daemon in question, the developer *should* respond in a timely and honest manner to requests from sysadmins. The 'real world' situation rarely reflects this ideal, however.

A recent example of a dysfunctional process is the code "fork" of the FOSS OpenOffice.org office suite into a separate product now named LibreOffice[86] – an action prompted by the purchase of Sun Microsystems Corporation (which had editorial and trademark control of OpenOffice.org) by Oracle Corporation. This created a perception that changes originating in the community were less likely to be accepted into the codebase [87, 88]. In this case, the free software community duplicated the OpenOffice.org codebase because Oracle's corporate culture, organisational priorities and interests appeared to clash with those of the community. Individual systems administrators, though, are unlikely to possess sufficient time, budget and ability to perform a similar code fork against a free software project whose log messages they would like to improve. This situation has led to literature (covered earlier) which presumes that event log messages effectively cannot be altered and must be accepted "as-is".

The key conceptual motivation here is: the developer (a corporation, designer, programmer, etc) is most often outside the organisation of the systems administrator and therefore does not perceive that organisation's priorities, policies, SLAs, and so on. Note that the word 'organisation' can be interpreted here as different units within the same entity. The pressures on the developer, and their motivations, are distinctly different from those of the sysadmin. By introducing a method for sysadmins to directly alter the contents of event log messages *without a patch or code fork*, a cycle of rapid iteration can be introduced; entirely within the sysadmin's organisation and consequently responsive to that organisation's particular requirements.

Current, 'normal' processes are shown in Figure 5.1. Sysadmins usually have a degree of control over the daemons they run, in the form of configuration options, which most often

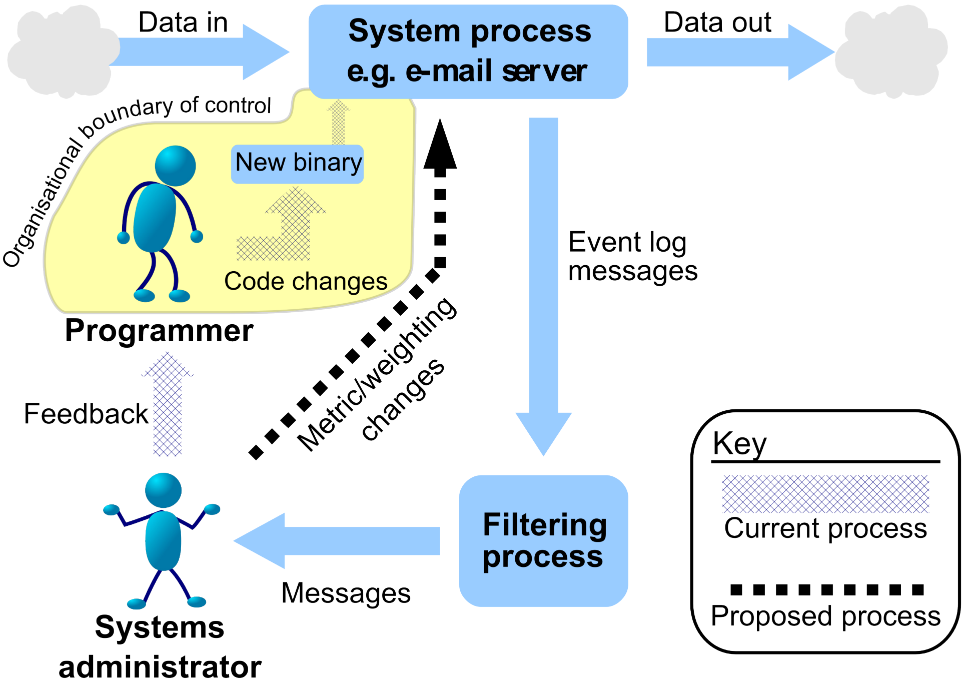


Figure 5.1: Model of rapid iteration for improving event log message output (programmer operating under the auspices of a separate organisation)

allow tweaking of log message verbosity or debug levels – offering the ability to 'strangle' log output to one extent or another. That is; verbosity and debug levels throttle the *number of messages* issued in given circumstances but do not allow the *messages' contents* to be altered or tuned. To perform any alteration, the sysadmin must provide feedback to the programmer/designer of the daemon or program in question, hoping that their feedback will be accepted and result in code changes in a future version of the daemon. Alternatively (and assuming a FOSS project), the sysadmin can effectively fork the source code of the project - by maintaining a code patch which changes messages to their liking. In the case of LibreOffice, external end-users and developers of the OpenOffice.org project forked the entire source code repository, duplicating it to form their own project. Such code-level changes are most likely the only way to modify natural-language messages simply because they cannot be mathematically/mechanistically transformed.

Bypassing the project's developer (as in the "proposed process" in Figure 5.1) simply mirrors the effect of the configuration options already provided by most daemons. An administrator whose systems are to accept e-mail for a new Internet domain name of course does not have to ask for a code alteration; the capability is provided in the Mail Transfer Agent's (MTA) configuration file. Elevating event log tuning to the same operational level as established capabilities is little more than an honouring of Buckley's call for logging to be taken seriously – for it to be seriously regarded as a feature rather than an afterthought [26]. Debug levels and verbosity settings are not a sufficient configuration mechanism after all - they merely provide a *passive* throttling mechanism. An ability to *actively* alter messages, pre-transmission, is needed to enforce organisational realities for event logging. The daemon is presumably only being run to facilitate organisational goals; why must its messages be otherwise?

The ability to iterate a configuration to match one's needs is hardly new. Autonomous stock market agents, however, have applied the technique to a data flow with common characteristics to those of event log messages, and their success has highlighted its usefulness when combined with the ability to make changes that actively alter the outcome. Introducing a mechanism for such tuning, i.e. customisation, is the focus of the next section.

5.3 Making iteration powerful, with weights

A mechanism for enforcing organisational realities on event log messages has one intended outcome: moderating notifications so that only events which can have an impact on the organisation are passed on. Each organisation would, of course, have different human thresholds and tolerances for "impact". For example: events on servers under test will not have a direct impact on a business simply because those machines are not in production use. Other cases are less clear-cut, such as whether a database failure will have more or less impact than a web server failure, and it is in these situations, especially when problem-solving resources are finite, that weights can become a useful expression of organisational priority.

Section 4.2 laid out the reasoning behind appropriating Mitchell et al.'s 'legitimacy' scale and re-naming it 'impact'. "Knowledge of organisational priorities for the event's component" was mentioned as one aspect of the scale - such a notion harks back to 'legitimacy' representing a wider scope than just one person's perception. Indeed, it incorporates the entire human context 'system' of norms, values and beliefs that exist in any organisation[61]. Yet our computer systems do not know where they fit in to this milieu, given their lack of those human attributes. Hierarchy, importance of self, quality (of its own hardware and software) - all these concepts are as foreign to a computing system as disappointment or satsifaction. The depth of a computer's knowledge of its own characteristics extends only as deep as kernel versions in software and motherboard model numbers in hardware. Only the systems administrator can think "that kernel build has been great", "I couldn't care less if this disk array failed" or "the whole lot is going to come crashing down if that jury-rigged network router dies", yet currently there is no mechanism for codifying such contextual information.

The basis for these perceptions of importance is easily deduced from the semiotic representational triangle (see Figure 2.2 on page 39, or the top-left of Figure 5.2); computers deal only with signs, humans are the observers/users, leaving the real-world objects - which are, of course, what humans find the most straightforward to base their thinking on and what we refer to when we use signs [2]. These same objects run the software created by developers and therefore are the originating point for our event log messages. Moreover, they (mostly) perform the roles assigned them by humans, such as DNS server, boundary router, or database server; a system's role determines its place in the hierarchy of *importance*, because the purpose it fulfills forms the relationship to organisational function rather than the piece of hardware or software itself.

Roles, however, are the abstract constructs of human minds and as such are merely embodied in particular pieces of equipment; whether virtual or physical, hardware or software. Then how can the role's organisational importance (which determines its ranking in the hierarchy of systems) be communicated? Especially given that the ranking is also an abstract construct (but like roles, still a 'real world object' at the social level). The application of the semiotics representational triangle to roles, rankings and weights is depicted in Figure 5.2, itself somewhat reminiscent of Barr's designer/user dichotomy (Figure 2.3 on page 42), to clarify the relationships between these terms.

The ranking can sometimes be thought of as the order in which systems should be restored in a disaster-recovery scenario. Simply put, conceptually embodying the ranking together with the software component allows a program/daemon to utilise the ranking for *weighting* purposes, i.e. elevating or depressing its own reports based on the ranking which represents human impressions of its importance. Weights, the expression of the ranking concept, seek only to modify and shape the programmer/designer's best anticipation for a unique computing environment: that of the organisation in which the program or daemon is running.

In keeping with the earlier call to enhance automation through the use of machine-readable metadata, the ranking must be expressed in a numeric form, i.e. a weighting. The alternative (that is, keywords) would limit granularity to pre-defined levels and re-introduce the ambiguity of RFC 5424 (see Table 2.1 on page 32) with its pre-assigned textual labels. The entire rationale for the weighting mechanism is that it allows organisations to represent their own unique hierarchy of importance, so it follows that the granularity should also be of their choosing. A floating-point number between 0 and 1 affords a transparent model wherein the seed value in the original event log message is simply multiplied by the weighting value to obtain a number which conveys the programmer's knowledge com-

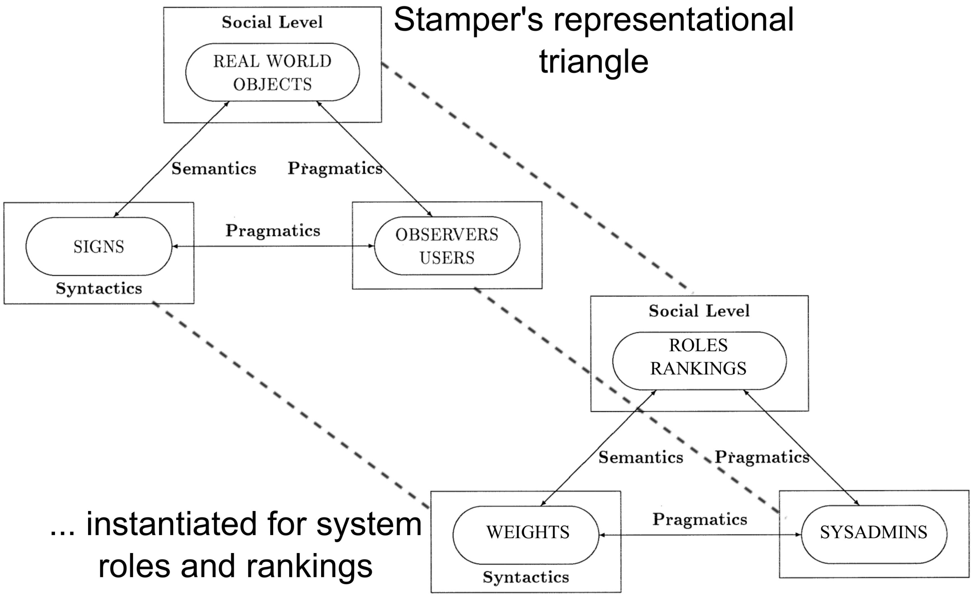


Figure 5.2: Semiotics theory [3] applied to roles, rankings and weights of information/-computer systems

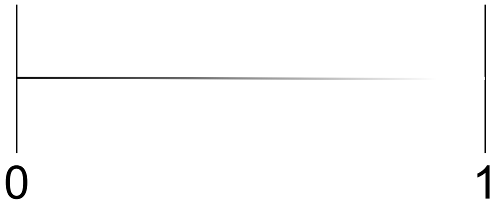


Figure 5.3: Floating-point weighting scale for representing roles and rankings as *organisational importance*

bined with organisational importance. Shown in Figure 5.3, “0” represents the extreme possession of importance, “1” represents the extreme lack of importance.

These weighting values can be altered without recourse to a code patch or writing a static-matching filter to match all the possible messages a program might produce. As with any other configuration option for a program/daemon, the setting can be experimented with in an iterative process. At this initial stage it seems appropriate for each daemon to accept one weighting simply because the daemon performs a role through service provision. In addition, the machine running the daemon should have its own weighting which is combined with all the impact values which pass through its Syslog forwarding system (i.e. forwarding event log messages to a central collection point). Situations such as a memory leak in a less-important daemon are therefore not obscured on an important server. The granularity for assigning weighting values must be based on the possible range of any influencing factor - the containment barrier, in effect - and this currently is based on physical or virtual hardware instances. Non Uniform Memory Access (NUMA) supercomputing clusters with a single memory address space, on the other hand, would apply the weighting to each partition of resources.

Organisations assign roles to their computing resources, which then assume a ranking in the hierarchy of importance, influenced by factors such as dependencies and business continuance. The 'impact' scope depicted in Figures 4.4 and 4.3 covers these areas by virtue of its focus on the larger environment surrounding any given system. Rankings within that scope can then be codified as *weighting* values; allowing their direct use in software to express and incorporate roles and rankings in a human- and machine-readable form. Scope and context also govern the number of weighting values to assign; the granularity of 'impact' is limited to replaceable components, i.e. daemons, programs, and the resource partitions they run within.

5.4 Summary

This chapter examined an identified parallel to event log messages in stock market high-frequency-trading algorithms. Rapid iteration of a code base is a critical capability for those carrying out such stock trading as a way of coping with market changes and the actions of competitors. It was then determined that a direct application of that particular technique would not be feasible with either open- or closed-source software. Configurable weights were then introduced as a method for achieving the goal of a convenient and low-maintenance method for altering event log output. Weights allow the organisational knowledge and priorities of the systems administrator to be incorporated into that output.

Chapter 6

Expressing a three-dimensional scale of salience

Inspired by the Glasgow Coma Scale (the GCS, a three-dimensional scale used globally for communication between medical professionals and triage [12]), a scale composed of ‘severity’, ‘impact’ and ‘certainty’ dimensions is proposed as a metric for communicating event log message salience. But how should each dimension be defined, at what granularity? The GCS incorporates the gravity of each dimension by using differing ranges: 1 to 4 for “eye response”, 1 to 5 for “verbal response” and 1 to 6 for “motor response”, but we must bear in mind its purpose before adopting such a technique (i.e. item scaling [ibid]). The GCS is intended as a tool with the following purposes:

Discrimination	Refers to the assessment of the depth of impaired consciousness and coma in patients with acute cerebral disorders and involves distinguishing severe from mild or moderate cerebral dysfunction.
Evaluation	Refers to the measurement of change in the level of consciousness of patients with cerebral dysfunction while under observation
Prediction	Refers to prediction of the outcome of these patients on the basis of their level of consciousness at the time of assessment

Table 6.1: Stated purposes of the GCS scale [12, pg. 755]

Any attempt to relate these purposes to the scopes of event log messaging is likely futile, but they should not be dismissed out of hand, either. ‘Discrimination’ can, for our purposes , be summed up as ‘severity’; ‘evaluation’ is equivalent to “rate of change” and ‘prediction’ is effectively “likely outcome” or “prognosis”. Some distinctions can immediately be seen. For example, no attempt has been made to incorporate any "rate of change" measurement into the event log messaging proposal here, as Syslog-recorded events tend to be short-lived points in time with no state to change; and "prediction" is similar in that it has been left out of this proposal given that event correlation researchers have demonstrated little to no success (in real-world situations).

Another blow to "prediction" is merely the pace of change in computing. The medical profession has developed slowly over millenia, yet just 150 years ago "bleeding" was still considered the go-to treatment despite its propensity for killing the patient [79]. Human anatomy has not changed appreciably over this timescale, apart from improved environmental conditions and nutrition bringing forth better overall health and taller populations. It is likely that there are more 'anatomical' changes to the Debian Linux "unstable" software repository in a single day than there have been to humans in thousands of years, to say nothing of the immense variation that might be contained in the umbrella terms "computer" or "operating system". Medicine is very often able to apply lessons that were learned many years prior, and is capable of making valid, confident predictions due to such a stable and unified (by comparison) human 'platform'. If computing had not changed appreciably in 40 years – i.e. the only computer available today was a Burroughs B5000 running the original, unmodified MCP operating system – the computer science discipline may have had greater success with event prediction and correlation.

Returning to "item scaling"; the GCS was designed for the dimensions to be added together to produce a score between "0" (dead) and "15" (normal function), so the relative importance of each dimension was factored in with numeric constraints on each one. More recent research has conclusively demonstrated the dimension-summing approach to be lacking in resolution - it actually discards useful dimensional information, resulting in poorer outcomes for patients in time-critical situations [12]. Reporting each dimension separately has become more common as a result.

The advantages of separated dimensions can easily be illustrated with event log messages, too. A disk fault in a RAID array may produce a log message prefix of "S:0 I:6 C:3" (for argument's sake) indicating high severity, low impact and middling certainty. To add these dimensions together would produce "9" from an event which has been catered for with the provision of redundant disks and is highly likely to be imperceptible for end-users. For comparison: a disk volume/partition fills up, preventing a volume snapshot from being created, and therefore an overnight backup process fails (which generates the event log message); this event may produce a prefix of "S:3 I:2 C:4" - indicating much less severity, greater organisational impact, but greater uncertainty. The dimensions still add up to "9" but for an event which hasn't been catered for and could even result in contractual or legal consequences. The other events in the chain would generate their own

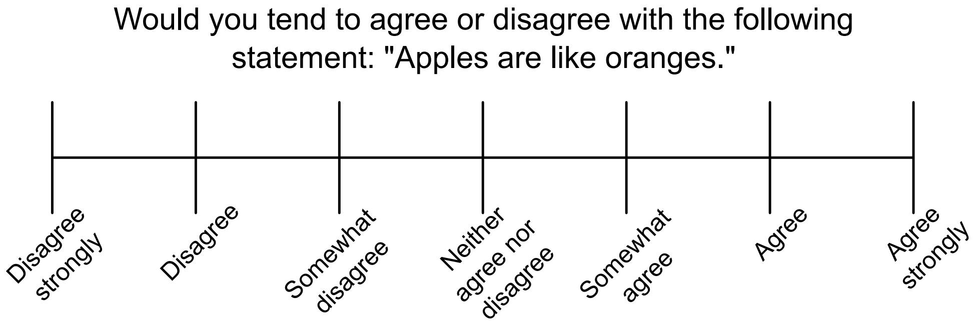

Would you tend to agree or disagree with the following statement: "Apples are like oranges."
Disagree strongly
Disagree
Somewhat disagree
Neither agree nor disagree
Somewhat agree
Agree
Agree strongly

Figure 6.1: Example Likert-type scale

messages and prefixes.

Reporting the dimensions separately allows discrimination between the three factors of event severity, organisational impact, and programmer/designer certainty. To combine them would result in one scale ranging from, for example: “0” (apocalypse) to “21” (inconsequential and uncertain), obscuring the source of the score, therefore limiting usefulness to events at either extremity. The ‘middle ground’ would be impossible to analyse as their scores could have come from any of the three dimensions. In essence a colour photograph would be converted to grayscale: red, green and blue become indistinguishable and luminance is all that remains.

6.1 Arguments for scale length and type

With seed values intended to be implemented in code (by the programmer/designer who is the individual best-informed about the algorithm and its assumptions), there is a valid argument that any given scale should conform to research conclusions on *response* scales. Such scales are meant to elicit unbiased responses from members of a population, as with surveys, and to dovetail with the cognitive nature of human beings when attempting to quantify opinions/attitudes [89]. That is; they offer two extremes and a mid-point. The ‘Likert-type’ scale (see Figure 6.1) typically does this with opinions/attitudes of preference or approval.

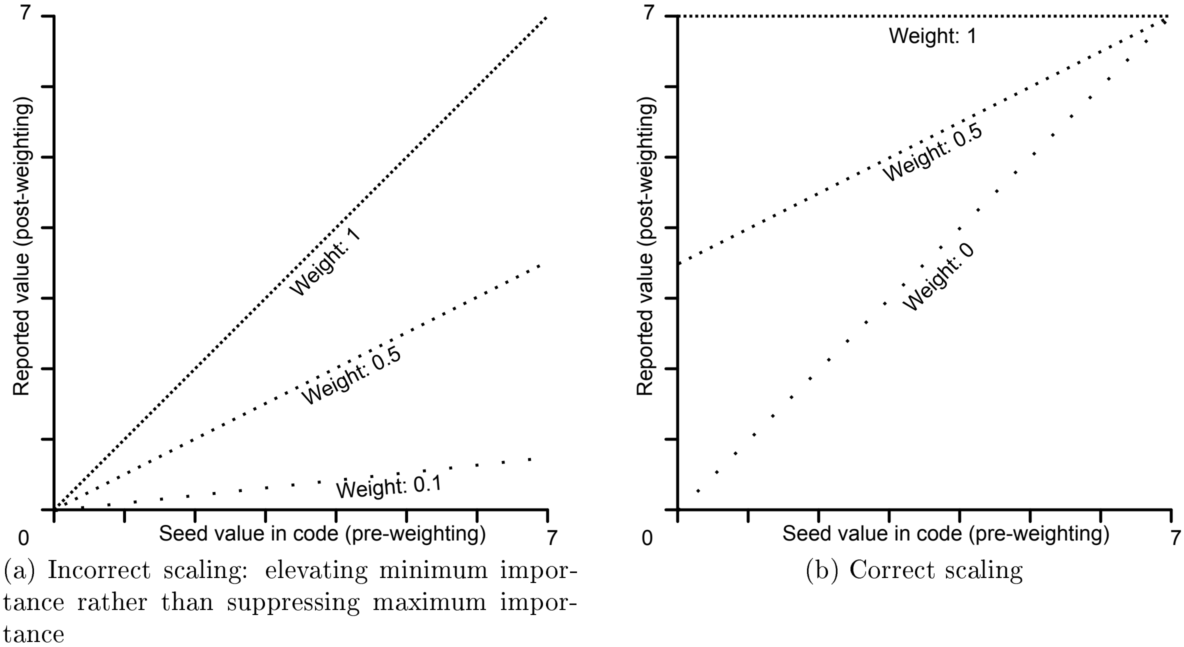
The underlying problem with applying response scales to the event logging context is this: they rely on sampling a natural population that will (with a large enough sample) return a bell-shaped standard distribution of results. The concept of the mid-point represents the “average” – and this is why the Likert-type scale, or similar ones such as the “linear-numeric” scale, consists of an odd number of response items (most often five or seven [89]). Event log messages do not conform to the standard distribution [10] nor can statistical techniques be used to predict them [26]. At the time of writing, they contain no usable and meaningful metric, either.

To propose a Likert or “linear-numeric” scale for an event logging metric would be to suggest that most messages should reside near the middle of the scale. Standard deviations and other statistical tools simply can’t be applied to a domain where the data much more closely resembles the Pareto principle (i.e. a few messages occur frequently while most messages occur rarely, but neither frequency nor ‘severity’ is any indicator of salience) [10]. It is therefore more logical to follow Buckley’s [26] advice and implement a linear scale calibrated by the extreme possibilities yet with no notion of a mid-point to imply a ‘norm’. The purpose here is not to dictate a ‘norm’ to developers or the organisations and their systems administrators that run daemons on their servers; but rather to allow the developers’ norms to be perceived and the organisation’s norms to be imposed on their running systems.

RFC 5424 is only the most recent RFC to document the Syslog severity scale (see Table 2.1 on page 32) but the scale itself has existed since the 1980s, with its first official documentation in 2001’s RFC 3164[90]. Encoded into three bits (to most efficiently pack it into one byte with five bits left for the logging ‘facility’), it features eight levels of severity numbered “0” through “7”. In reality there are undoubtedly millions of lines of software code, even when only counting FOSS projects, which already incorporate severity values according to RFC 5424. Simply discarding such an immense body of work, when those severity values could potentially alleviate by one-third the workload involved in implementing a three-dimensional logging scale, would be folly at best. This is not to deny that future work could evaluate the validity of the Syslog severity scale vs “linear-numeric” or another type. A priority in this work is to ease any possible integration into real-world software and re-using the Syslog severity scale dovetails with that aim. Moreover, it features suitable extremes, no explicit notion of a mid-point, and what shall next be explained as a feature: beginning with “0”. A disadvantage is that its ordering requires a marginally more complex weighting function.

6.2 The application of organisational salience

At first sight, it appears that any numeric scale must start with a “0” value representing the least important end, in order to incorporate weightings (see Section 5.3 on page 92) via fractional (0...1) multiplication, and this is correct. The RFC 5424 scale, however, regards “0” as the most severe extreme - meaning that weighting with fractions produces the inverse of the intended outcome. That is, the scaling ‘asymptote’ resides at zero when it is meant to be “7”, the least severe extreme, because the mechanism of weighting is primar-



(a) Incorrect scaling: elevating minimum importance rather than suppressing maximum importance

(b) Correct scaling

Figure 6.2: Weighting results graphs

ily meant as a straightforward method for suppressing spurious/unimportant messages (where "spurious" or "unimportant" is defined by the organisation running the program or daemon). In effect, the least important events would be elevated in importance. This scenario is illustrated in Figure 6.2a.

As can be seen in Figure 6.2b, the 'correct' scaling has the effect of suppressing the reports from components judged less important. A system or daemon with a weighting of "1" will always report its events as being of minimum impact while a weighting of "0" results in no suppression. Whether the scaling should be characteristic of a straight line or not across all seed values in the code would be a worthwhile topic for future research. A parameter for specifying a curved scaling line could certainly be devised, perhaps at the cost of introducing some complexity and administrator uncertainty about the results.

Another research question regarding the weights might be: what distribution are the weights themselves likely to form? If we accept, for argument's sake, that an organisation might define the mid-point of "0.5" as the appropriate weighting for the archetypal "average system", then is it possible or likely for the distribution of weights to resemble a standard distribution? Such a distribution would permit the usage of diverse existing theories and analysis tools. It would represent the combined assessments of importance for messages issued by all the source components included in it, regardless of its shape, meaning the application of the weighting algorithm is almost inevitably going to transform the Pareto-like distribution of event log messages into an as-yet unknown form.

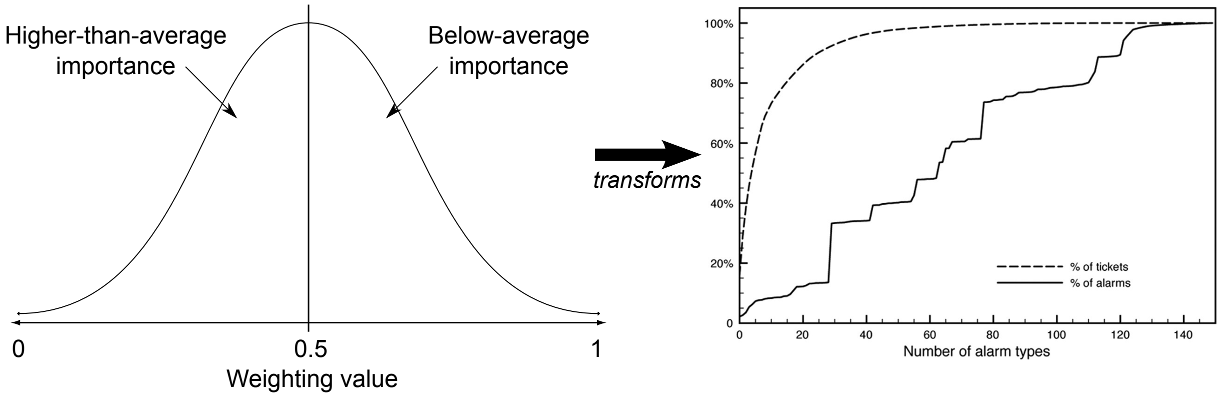


Figure 6.3: Process of applying a normal distribution of weights to Wallin et al.'s Pareto-like distribution [10, pg. 10] of event alarms & tickets; the result is unknown (see text)

Note: the alarms graph extends out to 3500 alarm types [10], indicative of the 'long tail' in this problem domain.

Depicted in Figure 6.3, the application of weights (which are per-message-source, reflecting the importance of each component in the organisation's context) to the previously defined three-dimensional scale would likely produce a significantly different distribution of event log messages. It is important to remember that the number of message types will not change as the output will not be 'throttled'; Wallin et al.'s alarms domain, with only 3,500 unique types, more closely resembles the less packed space of the Munin monitoring tool (see Table 3.1 on page 72) than more general event logs with higher ratios for Weaver's *H*. Instead, the change will be in the plotting of the "% of tickets" dashed line as a result of *source importance* being incorporated via scaling. Possible outcomes are as yet unknown since trials would have to be run in valid production environments to gather sufficient data. Such an experiment would entail the patching of source code, compilation of patched programs/daemons, and the assignment of an importance weighting to all the participating components. The outcome mockup presented later does not feature enough data points to demonstrate any transformation.

The scaling function mentioned up to this point was originally intended to be a simple multiplication of the seed value in code and the source's weight. The re-use of the Syslog severity scale then required a reversal of the scaling function (as explained above). It remains a simple function, resulting in the graph shown in Figure 6.2b.

Algorithm 6.1 Simple scaling function for suppressing the perceived impact of messages from less important sources. Calibrated to 'Syslog severity' extremes.

$$i_{reported} = i_{seed} + w(7 - i_{seed})$$

With “i” as the ‘impact’ metric and “w” as the per-source weighting value, the scaling function allows the contextual role of the source system component to be reflected in all its event log messages, even those messages which have never before been issued in a given organisation. The mechanism avoids any delays attributable to the achieving of statistical significance or a message being placed in the large bin of “unknowns”, as happens today with static-matching solutions. On the other hand, it relies on systems administrators or policy-makers for the assignment of sensible and useful weights, and the programmer/designer for sensible and useful seed values; neither of these caveats are new, however, as existing ‘solutions’ place similar burdens on these people today.

6.3 Outcome mockup

To illustrate some of the possible outcomes with the presented design, selected event log messages will be shown with prefix values assigned by the author. The prefixes take the form of a `[S?,I?,C?]` tuple (hereafter called a *SIC tuple*) to communicate each message’s rating according to its severity, certainty and impact. For example, the message “`kernel`*[S4,I6,C4]*`: ATM dev 0: error -110 fetching device status`” has been assigned a severity of “4”, an impact of “6” and a certainty of “4”. The mockup scenario includes three server systems representative of several different organisational roles:

1. Server A: A critical firewalling, proxying and external e-mail system, essential for organisational activities.

2. Server B: An internal server running network authentication daemons, a few internal databases of average importance, and some internal e-mail storage. Some of this system’s workload can be handled by other machines running redundant services for fail-over purposes.

3. Server C: An internal web site development machine which is 90% used for testing and development rather than production sites. Beta sites may occasionally be made available to a small, selected audience.

Three examples from each server were selected for the sake of brevity and the reader’s benefit, and can be seen in Listing 6.1. The complete data set is contained in Appendix B on page 155, as Listings 10.1, 10.2 and 10.3 (containing 28 messages each). The messages were chosen, using the author’s experience of systems administration, from a genuine flow of event log messages which accumulated 32.7MB of plain-text data, to demonstrate a

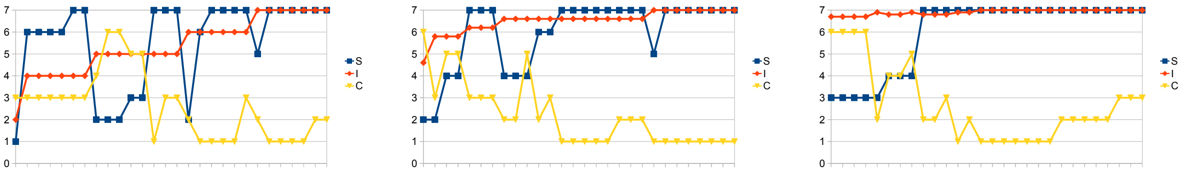
Server A			Server B			Server C		
Impact weight: 0			Impact weight: 0.6			Impact weight 0.9		
S	I	C	S	I	C	S	I	C
6	4	3	7	6.6	2	7	6.8	2
3	5	5	4	5.8	5	7	7	1
6	4	3	4	5.8	5	7	6.9	1
6	4	3	7	7	1	7	7	1
2	5	6	7	6.6	1	7	7	1
3	5	5	7	6.2	3	7	7	1
2	5	4	7	7	1	7	7	1
6	4	3	7	6.2	3	7	7	1
2	5	6	7	7	1	7	7	1
5	7	2	4	6.6	5	7	6.8	2
7	4	3	6	6.6	3	4	6.8	4
7	5	3	7	6.2	3	4	6.8	4
7	5	3	7	7	1	7	7	3
7	6	3	7	7	1	7	7	3
7	6	1	4	6.6	2	7	7	3
7	4	3	4	6.6	2	3	6.7	6
7	7	1	7	6.6	2	3	6.7	6
7	7	1	6	6.6	2	3	6.7	6
7	5	1	7	6.6	2	3	6.7	6
7	7	1	7	6.6	1	7	7	2
7	7	2	7	6.6	1	7	7	2
7	7	1	7	6.6	1	7	7	2
7	7	2	7	7	1	7	7	2
7	6	1	7	6.6	1	7	7	2
1	2	3	7	7	1	3	6.9	2
7	6	1	5	7	1	4	6.9	5
2	6	2	2	4.6	6	7	6.8	3
6	6	1	2	5.8	3	7	6.9	2

Table 6.2: Summary of outcome-mockup values, extracted from the contents of Listings 10.1, 10.2 and 10.3 in Chapter 10 on page 155

Note: the “I” value given here is the reported impact derived from the seed impact value and the configured weighting for each system being monitored.

Listing 6.1 Mocked-up event log messages (selected examples)

```
Jan 30 06:56:56 serverA postfix/smtpd[31944][S6,I4,C3]: disconnect from unknown
    [190.51.227.124]
Jan 30 06:56:56 serverA postfix/smtpd[31944][S2,I5,C6]: lost connection after DATA (0
    bytes) from unknown[190.51.227.124]
Jan 30 07:45:58 serverA ntpd[2304][S5,I7,C2]: kernel time sync status change 0001
...
Feb  1 19:10:15 serverB krb5kdc[2656][S6,I6,C2]: AS_REQ (3 etypes {16 1 3}) 172.16.1.100:
     NEEDED_PREAUTH: example@EXAMPLE.COM for krbtgt/EXAMPLE.COM@EXAMPLE.COM, Additional
    pre-authentication required
Feb  1 19:10:15 serverB krb5kdc[2656][S7,I6,C2]: TGS_REQ (3 etypes {16 1 3})
    172.16.1.100: ISSUE: authtime 1296540615, etypes {rep=16 tkt=16 ses=16},
    example@EXAMPLE.COM for host/serverB.example.com@EXAMPLE.COM
Feb  1 19:25:19 serverB cyrus/ctl_cyrusdb[31955][S7,I6,C1]: archiving database file: /var
    /lib/cyrus/mailboxes.db
...
Feb  6 12:28:53 serverC apache2[8472][S4,I5,C4]: [06/Feb/2011 12:28:52 20583] [error]
    OpenSSL: error:1407609C:SSL routines:SSL23_GET_CLIENT_HELLO:http request [Hint:
    speaking HTTP to HTTPS port!?]
Feb  6 13:50:57 serverC kernel: [511040.208020] hub 1-1:1.0[S7,I7,C3]: activate --> -19
Feb  6 13:50:57 serverC kernel: [511040.208092] usb 1-1[S7,I7,C3]: USB disconnect,
    address 3
```



(a) Server A; impact weight "0" (b) Server B; impact weight "0.6" (c) Server C; impact weight "0.9"

Figure 6.4: Outcome-mockup values sorted by **I**mpact, then **S**everity, then **C**ertainty

range and variety of SIC tuple values that would occur in real systems, while avoiding copious numbers of repetitive messages.

Summarising the tuples from the mockup yields the information shown in Table 6.2 on the facing page, after incorporating a per-system weight for the impact column. For example, the Server B event for `cyrus/ctl_cyrusdb` at 19:25:19 on February 1 has a seed impact of "6" and server B has a weighting of "0.6", so the reported impact (using Algorithm 6.1) is "6.6". When graphing these values, as can be seen in Figure 6.4 for the purpose of comparison, each line is sorted first by Impact, then by Severity within each Impact bracket, then by Certainty within the two preceding brackets. The effect of the simple weighting algorithm can be observed in the 'impact' plot line curve.

Consolidating event log messages from (potentially) many systems is of course the goal here and the combined graph is in Figure 6.5 on the next page. The horizontal axis shows a fairly even distribution of impact values from the most important system (server A) to the least important (server C). Another point to note is the wide variation of the severity and certainty values with respect to impact; no stable relationship appears until impact and severity both reach "7". At this point, only 'certainty' can vary, whereas earlier no

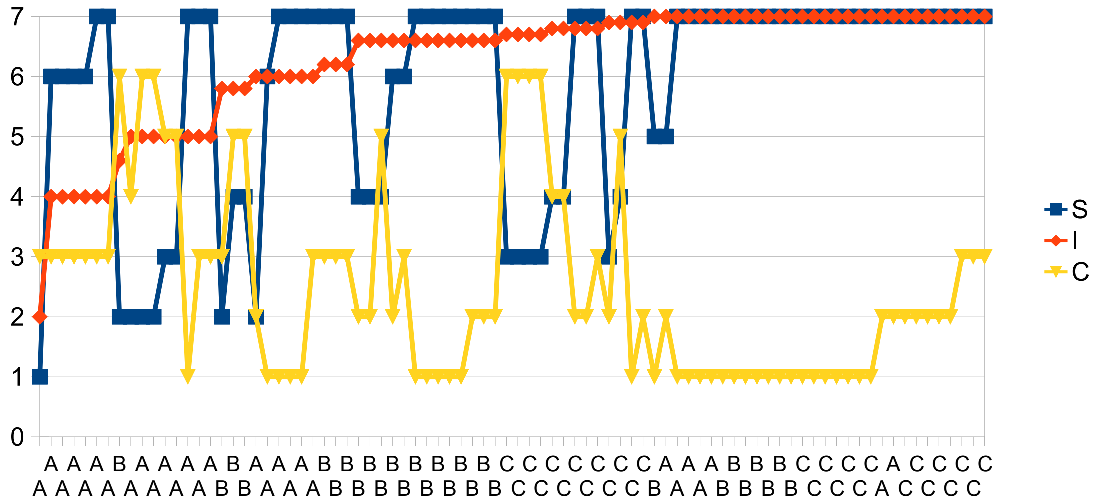


Figure 6.5: Consolidated outcome mockup values, sorted by **I**mpact, then **S**everity, then **C**ertainty

clear correlation appears between the three variables. This finding is closely aligned (for severity and impact) to Wallin et al.'s determination that severity correlates poorly with real-world priority [10].

With the three-dimensional metric in place, several simple `if/then/else` conditionals could be used to apply organisational policies (e.g. Service-Level-Agreement-specified responses) based on considerations of each dimension. For example, an `if` statement might page a systems administrator if `severity` $\leq$ `2`, `impact` $\leq$ `2` and `certainty` $\leq$ `4`. This statement would automatically activate for only one message in this example: "`+3.3V: +3.12 V (min = +3.13 V, max = +3.45 V)`", which shows one of server A's motherboard voltages outside of its acceptable range. The same message on servers B or C would *not* page the administrator because those machines are not considered as important; naturally another conditional could be written to pick up such an event on B or C (e.g. `severity` $\leq$ `2` and `certainty` $\leq$ `4`) and add it to a daily e-mail report. It is likely that 5-10 conditional statements could handle many environments, while also handling messages which were not anticipated, and yet *not* require the regular maintenance that existing solutions do.

6.4 Summary: outcome mockup conclusions

The capabilities demonstrated with the mockup appear to be unique; namely the notions of organisational importance and policy implementation (as those notions were presented in a NASA paper by Schreckenghost et al., [45], although that effort focussed on HCI etiquette for information presentation to astronauts[1]). A similar set of capabilities with a filtering solution utilising regular expressions would require the maintenance of separate sets of regexes for each and every server. No policy or organisational-importance capability was noted in any of the literature reviewed on data mining or other statistical approaches (including neural networks). Could data mining approaches incorporate policies? Would such algorithms require training for each operational profile or server? These topics would be worthy avenues for research in that direction.

The graphs produced also open up possibilities for graph analysis to establish a 'nominal state' that additionally takes the volume of events into account. Significant data volumes can be looked at in retrospect, when the state of a server has been confirmed as "good", and tolerances assigned to its graph output, leading to prompt detection of operation outside of these established norms. Each server would have its own "profile" represented by the graphs of SIC tuples it produces and this profile is likely to change with extensive modifications to the system such as an operating system upgrade. Once the systems administrator was satisifed that a sufficient amount of 'nominally-good' log data had been collected, the profile graph could be regenerated. As for what could be protected against with this approach, one example would be a volume-based attack such as a distributed-denial-of-service or dictionary attack, which would quickly distort the shape of the SIC-metric graph. This is not a "new is bad" technique: the metric portrays the salience of the information, not its raw volume or newness.

'Leaky buckets' are another mechanism in this regard: a rate limit (e.g. 200 per minute) of messages with a certain SIC metric, when exceeded, could trigger actions within seconds of an attack beginning. These possibilities will not be examined further here but are left as open questions for future efforts. It is also acknowledged that they might inherit some of the disadvantages of statistical approaches; principally, a time lag between initial detection and sufficient significance being achieved to warrant action being taken. The advantage of the presented approach is the machine-readable metadata which can inform such algorithms.

[1] Where 'etiquette' refers to the need to follow established conventions such as chains of command and sleep schedules, i.e. taking into account the larger context outside the computer system.

Chapter 7

A survey of systems administrators

Throughout this thesis the paucity of academic research into the situation of systems administrators has frequently been raised. In short, it appears that these software and hardware users has been regarded as the 'expert' population and therefore not in need of help when it comes to interfacing with computing technology. Their level of systems knowledge would surpass that of the academic community in many or most cases. Evidently, though, complexity and information overload is catching up with our coping abilities and techniques [29, 15, 91, 58, 59, 62, 10, 38, 18].

A lack of research into the opinions or priorities of sysadmins has led to anecdotal evidence being cited and unjustified thresholds being used, as demonstrated in Wallin et al. [10] and Yamanishi & Maruyama [18]. To an extent, this is understandable: systems administrators are often under very significant pressure given their responsibility for an organisation's computing infrastructure. There is no one industry body with membership being legally obligated, as is the case with the medical and legal professions. There isn't even a centralised ethical standard by which systems administrators must abide. Such is the state of a profession which only emerged within the last few decades and is perpetually in flux as the industry changes (e.g. due to outsourcing).

7.1 Demographics

The New Zealand Network Operators Group's (NZNOG) [92] annual conference in January 2011 presented an opportunity to conduct such a survey. NZNOG is not an organisation *per se*, but rather a mechanism for network, Internet-Service-Provider, and systems administrators to communicate with each other and collaboratively work through challenges facing themselves and the industry, primarily through a mailing list. The annual conference is aimed at developing face-to-face relationships and disseminating knowledge about upcoming products, Internet standards, Internet governance, and the like. The NZNOG membership is highly technical in nature and not affiliated with any academic goals or institution. Participation in the conference is entirely voluntary (although restricted to registered attendees); as a result the audience is likely to vary from day to day, due to external commitments.

The audience was therefore a mix of systems administrators, network administrators, ISP staff (often overlapping with the previous two categories), telecommunications workers, managers, marketing staff and vendor representatives. It cannot be overemphasised that these people are usually difficult to gain access to, and are often under great pressure, due to their pivotal roles in their organisations [10]. NZNOG has a tradition of technical discussions held under 'Chatham House Rules', wherein any details given that may betray competitive plans or technology are held in confidence, and are never to be attributed to the speaker. An entirely anonymous survey that does not solicit any identifying information is therefore the appropriate choice for facilitating a free and fair discussion.

7.2 Procedure

The survey was administered following a preliminary 25-minute presentation of this research. These actions were carried out in a hotel conference room, beginning at 11:45 on a Friday morning; the second day of presentations made at the conference. Responses were collected as the participants broke for lunch after a subsequent, unrelated presentation; the author held out a large labelled box and thanked participants as they returned the forms. The chaotic pile of forms was then shuffled, checked for any identifying information such as names (there were none) and stored. The box was left in place, unobserved by the author, until the end of the lunch break to allow any further responses; five were received in this manner. As shown on the form itself in Figure 7.2, the forms will be held securely for one year and then disposed of. Destruction will be via shredding.

The audience numbered roughly 120: 50 responses were received, giving a response rate of around 42%. This is higher than the average organisational response rate of 35.7% (std. dev. 18.8) reported by Baruch & Holtom, in 2008, [93] for surveys. As noted above regarding the demographics surveyed, the audience did not consist entirely of systems and network administrators or people with equivalent experience.

7.3 The survey instrument

The research goals for the survey were aimed largely at deducing what kind of event log information systems administrators found to be salient. Secondary to this goal were the desires to establish the prevalence of event log filtering, the techniques used and the effectiveness of those techniques. Validation/support for the concept of the three-dimensional scale (severity, impact, certainty) was also sought.

The instrument used is shown in Figures 7.1 and 7.2. It was printed on both sides of one sheet of A4 paper. Alreck & Settle's resource *The Survey Research Handbook* [89] was heavily consulted during the creation process. Linear-numeric scales were used for three items where the average response was expected to be around the scale's mid-point. One notable omission only realised after the survey had been completed was the lack of calibration information for the rankings item (number seven on page two of the form). Around 18% of respondents seemed to reverse the order of the Syslog scale and used "10" as the most important extreme when that was intended to be "0" (as was explained in the presentation immediately beforehand).

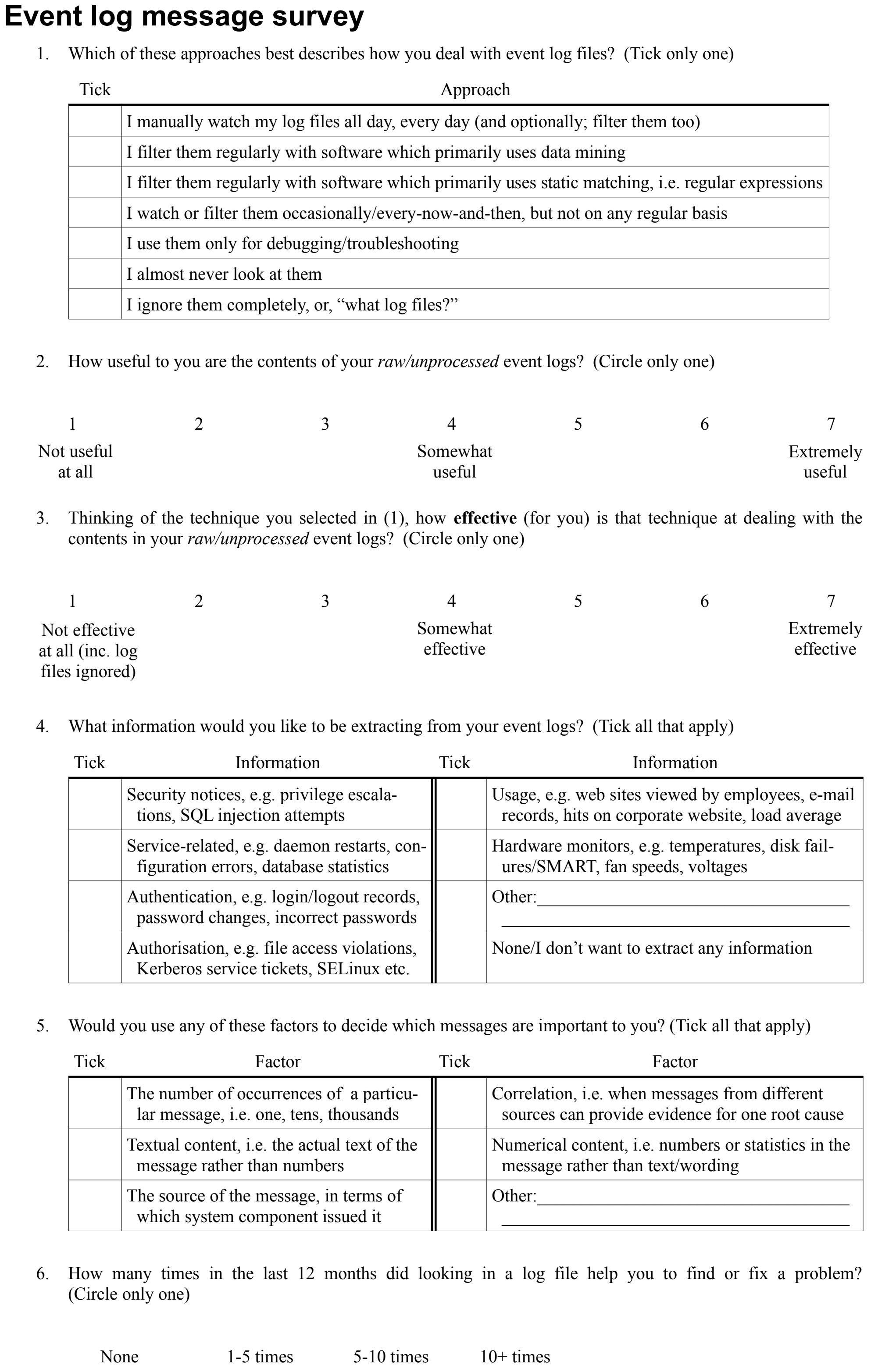
The vast majority of *those* responses appear to have reversed both extremes, rating items such as the UPS check - an entirely innocuous message simply noting a successful self-test - as "0" and the disk fault, a failure likely to cause immediate and severe problems for the system, as "10", exactly the opposite of most other responses[1]. The Syslog scale, as explained on page 32, regards "0" as the most important, and "7" as the least important. The survey used "1"-"10" merely as ranks.

This two-page response form was approved under Section 4.7 of the Victoria University of Wellington Human Ethics Policy [94], avoiding the need for a formal Human Ethics Committee approval process. Section 4.7 sets out conditions that were taken into account when designing the instrument, and the relevant excerpt can be viewed in Chapter 11 on page 161.

7.4 Raw survey results

The raw results are presented in tabular form in Chapter 12 on page 163. The size of the data set requires a multi-page format, with each response spread over two pages. One duplicate was noticed: form 28 appeared to be an exact facsimile of form 27, right down to the handwriting. Form 28's results were therefore omitted.

[1] See Section 7.5.3 for a discussion on the data cleaning process that was carried out.



Event log message survey

1. Which of these approaches best describes how you deal with event log files? (Tick only one)

Tick	Approach
	I manually watch my log files all day, every day (and optionally; filter them too)
	I filter them regularly with software which primarily uses data mining
	I filter them regularly with software which primarily uses static matching, i.e. regular expressions
	I watch or filter them occasionally/every-now-and-then, but not on any regular basis
	I use them only for debugging/troubleshooting
	I almost never look at them
	I ignore them completely, or, “what log files?”

2. How useful to you are the contents of your *raw/unprocessed* event logs? (Circle only one)

1	2	3	4	5	6	7
Not useful at all			Somewhat useful			Extremely useful

3. Thinking of the technique you selected in (1), how **effective** (for you) is that technique at dealing with the contents in your *raw/unprocessed* event logs? (Circle only one)

1	2	3	4	5	6	7
Not effective at all (inc. log files ignored)			Somewhat effective			Extremely effective

4. What information would you like to be extracting from your event logs? (Tick all that apply)

Tick	Information	Tick	Information
	Security notices, e.g. privilege escalations, SQL injection attempts		Usage, e.g. web sites viewed by employees, e-mail records, hits on corporate website, load average
	Service-related, e.g. daemon restarts, configuration errors, database statistics		Hardware monitors, e.g. temperatures, disk failures/SMART, fan speeds, voltages
	Authentication, e.g. login/logout records, password changes, incorrect passwords		Other:____________________ ________________________
	Authorisation, e.g. file access violations, Kerberos service tickets, SELinux etc.		None/I don’t want to extract any information

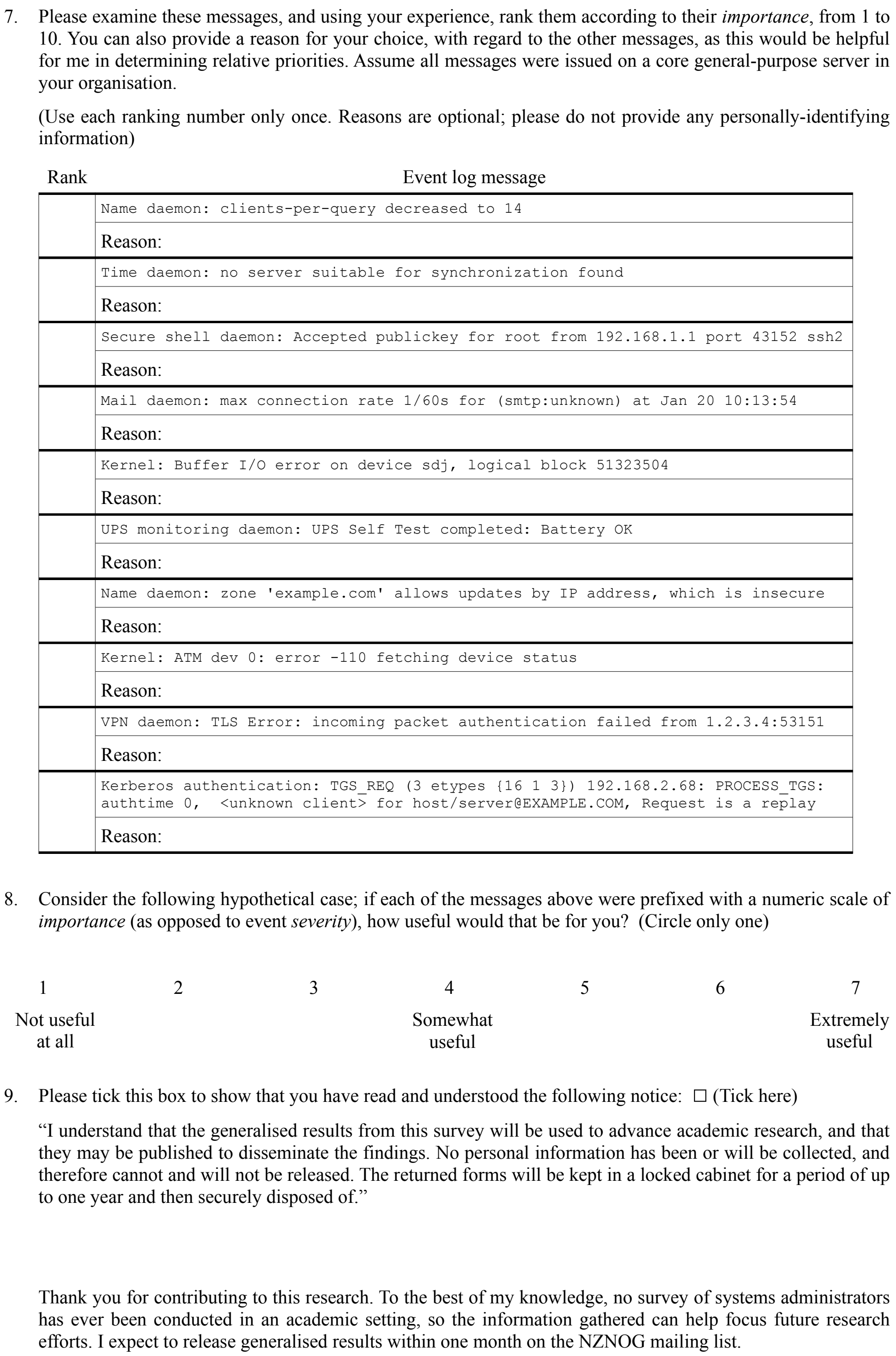
5. Would you use any of these factors to decide which messages are important to you? (Tick all that apply)

Tick	Factor	Tick	Factor
	The number of occurrences of a particular message, i.e. one, tens, thousands		Correlation, i.e. when messages from different sources can provide evidence for one root cause
	Textual content, i.e. the actual text of the message rather than numbers		Numerical content, i.e. numbers or statistics in the message rather than text/wording
	The source of the message, in terms of which system component issued it		Other:____________________ ________________________

6. How many times in the last 12 months did looking in a log file help you to find or fix a problem? (Circle only one)

None 1-5 times 5-10 times 10+ times

Figure 7.1: Survey instrument, page 1



7. Please examine these messages, and using your experience, rank them according to their *importance*, from 1 to 10. You can also provide a reason for your choice, with regard to the other messages, as this would be helpful for me in determining relative priorities. Assume all messages were issued on a core general-purpose server in your organisation.

   (Use each ranking number only once. Reasons are optional; please do not provide any personally-identifying information)

Rank	Event log message
	`Name daemon: clients-per-query decreased to 14` Reason:
	`Time daemon: no server suitable for synchronization found` Reason:
	`Secure shell daemon: Accepted publickey for root from 192.168.1.1 port 43152 ssh2` Reason:
	`Mail daemon: max connection rate 1/60s for (smtp:unknown) at Jan 20 10:13:54` Reason:
	`Kernel: Buffer I/O error on device sdj, logical block 51323504` Reason:
	`UPS monitoring daemon: UPS Self Test completed: Battery OK` Reason:
	`Name daemon: zone 'example.com' allows updates by IP address, which is insecure` Reason:
	`Kernel: ATM dev 0: error -110 fetching device status` Reason:
	`VPN daemon: TLS Error: incoming packet authentication failed from 1.2.3.4:53151` Reason:
	`Kerberos authentication: TGS_REQ (3 etypes {16 1 3}) 192.168.2.68: PROCESS_TGS: authtime 0,  <unknown client> for host/server@EXAMPLE.COM, Request is a replay` Reason:

8. Consider the following hypothetical case; if each of the messages above were prefixed with a numeric scale of *importance* (as opposed to event *severity*), how useful would that be for you? (Circle only one)

1	2	3	4	5	6	7
Not useful at all			Somewhat useful			Extremely useful

9. Please tick this box to show that you have read and understood the following notice: □ (Tick here)

   "I understand that the generalised results from this survey will be used to advance academic research, and that they may be published to disseminate the findings. No personal information has been or will be collected, and therefore cannot and will not be released. The returned forms will be kept in a locked cabinet for a period of up to one year and then securely disposed of."

Thank you for contributing to this research. To the best of my knowledge, no survey of systems administrators has ever been conducted in an academic setting, so the information gathered can help focus future research efforts. I expect to release generalised results within one month on the NZNOG mailing list.

Figure 7.2: Survey instrument, page 2

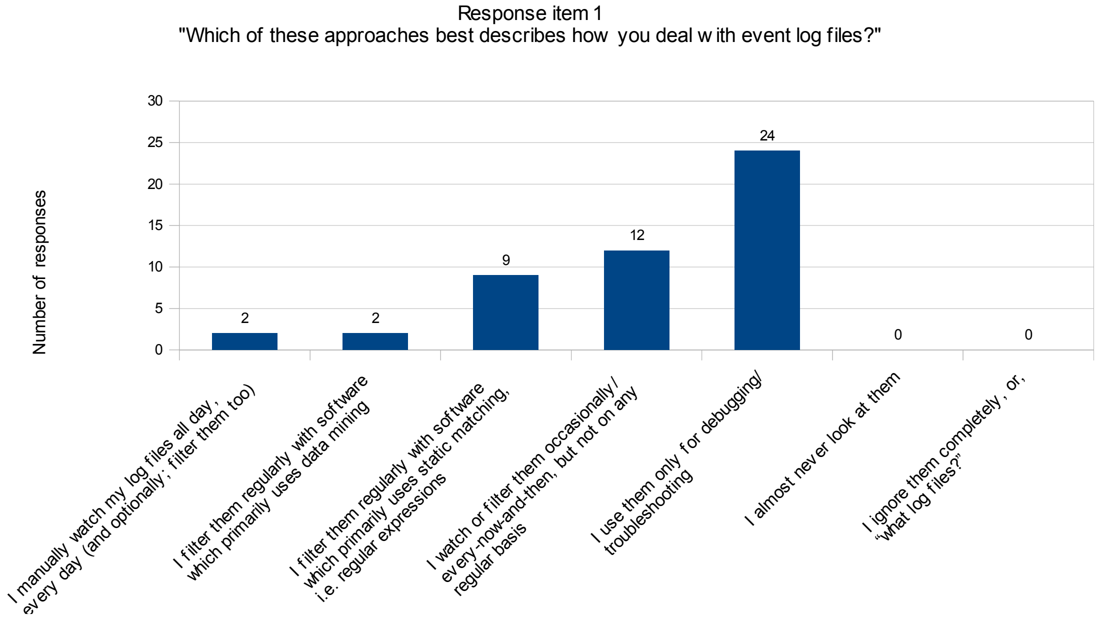


Figure 7.3: Response item 1 results

7.5 Analysis of results

7.5.1 Techniques in use

Figure 7.3 shows us that the most common 'technique' for dealing with log files is simply to leave them alone until they are needed after a specific event. That is; they are reserved for reactive use instead of proactive interpretation. Only 27% of participants regularly monitored or filtered their log files. Notably, static matching techniques are over four times more likely to be employed than data-mining.

An impression of 'comfort' with the existing situation can be garnered from Figures 7.4 and 7.5. Both data sets have medians of four, and means significantly towards the right: "4.43" for item 2, "4.35" for item 3. The audience in general appears somewhat ambivalent to raw event logs with the most-expert feeling right at home. Certainty about the effectiveness of their chosen 'coping method' for log files, however, drops off rapidly above "5", with comparatively few participants willing to state unreservedly their belief in the method they employed. One might posit that this reflects an awareness that much of the information generated at system level is simply never examined, but that conclusion must be considered in light of the survey being held immediately after the author's presentation about the issues created when using event log messages for system monitoring.

The vision presented in Figure 7.6 on page 116 (mean rated effectiveness for each technique) incorporates averages with as few as two data points and this should be kept in mind. An important conclusion is the small range within which the means reside; from

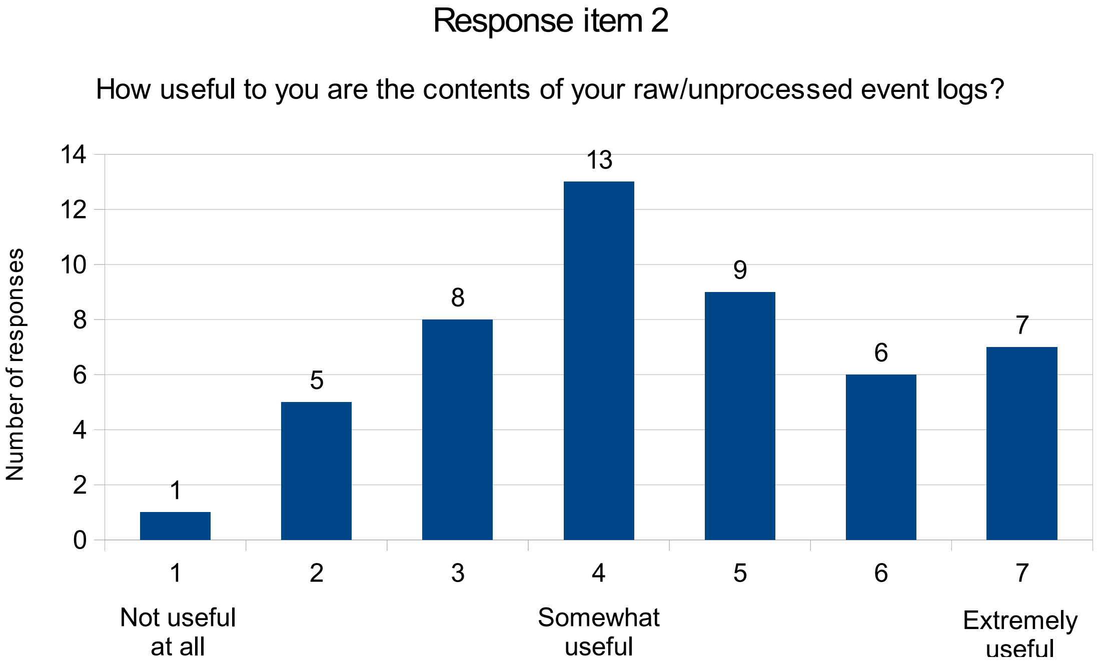


Figure 7.4: Response item 2 results

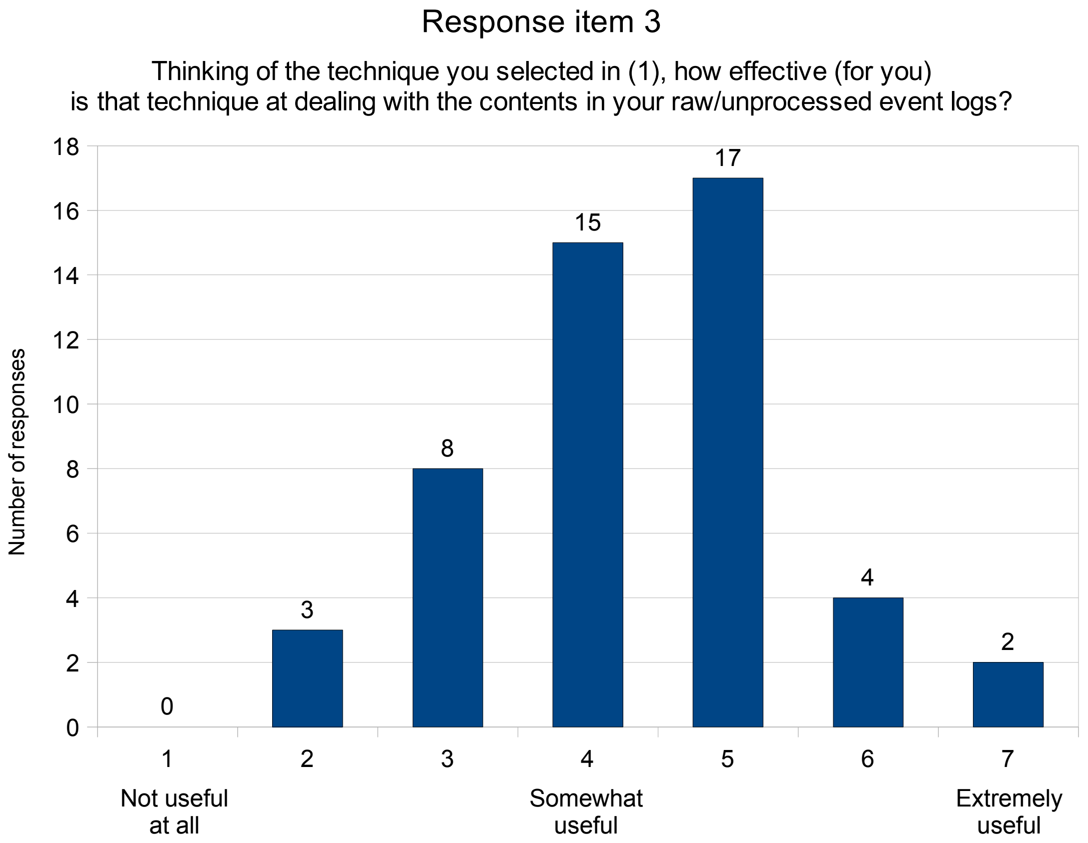


Figure 7.5: Response item 3 results

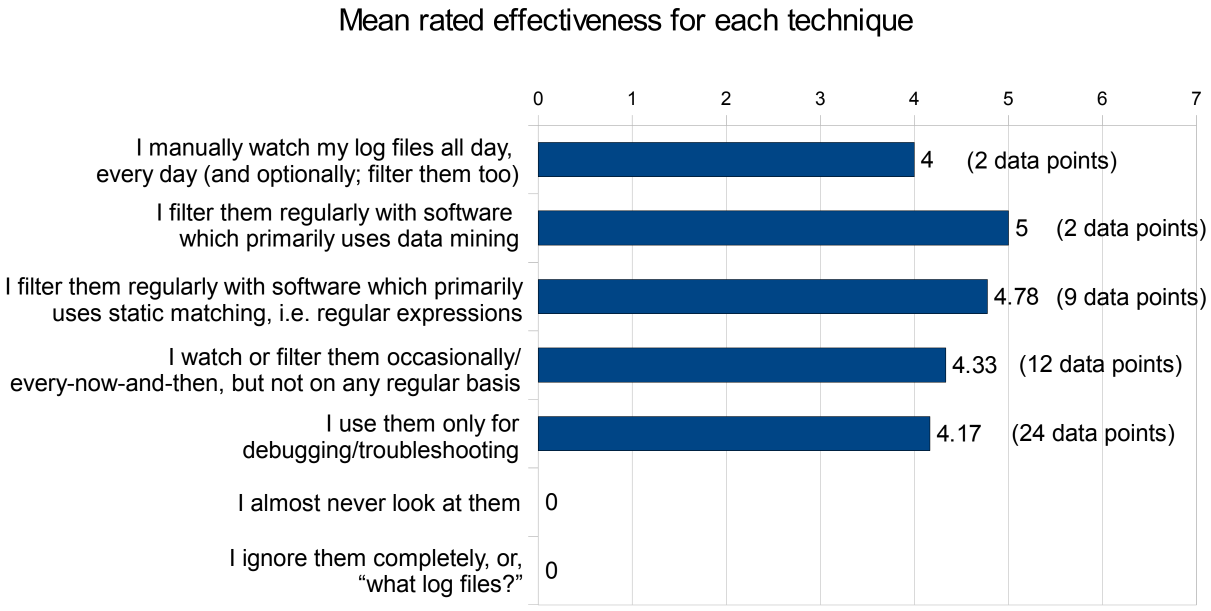


Figure 7.6: Mean rated effectiveness for each technique

"4" to "5" for effectiveness of each technique. This certainly does not suggest any great dissatisfaction (on average) with the choices that systems administrators have made, but neither does it display great satisfaction.

7.5.2 Motivations

Figure 7.7 shows the 'desire' of sysadmins to monitor their systems more closely; only one participant indicated that they did not have any interest in log file information. Note that "kernel messages" was not included as an option, as messages from the kernel are typically treated according to cause, such as disk failures under "hardware monitors" and stack-smashing attacks under "security notices". The two most heavily-favoured response items were "service-related" (37) and "hardware monitors" (34) - both of which are critical to *service availability*, and therefore potentially require the *most urgent* response from a systems administrator; if a provided service (e.g. e-mail, web proxy) or hardware component (e.g. ISP border router) is no longer available, there can be an *impact* on those who rely on 'our service'.

The other two response items in the majority were "security notices" (33) and "authentication" (30). Neither of these are directly related to immediate service availability (except perhaps via individual account compromises or corrupt/deleted data after-the-fact) and are closer to the concept of *system integrity*, i.e. maintaining a consistent and known state without rogue actors or processes. System integrity creates trust in the system's output; which will be deterministic in nature. For example, a serious security notice with

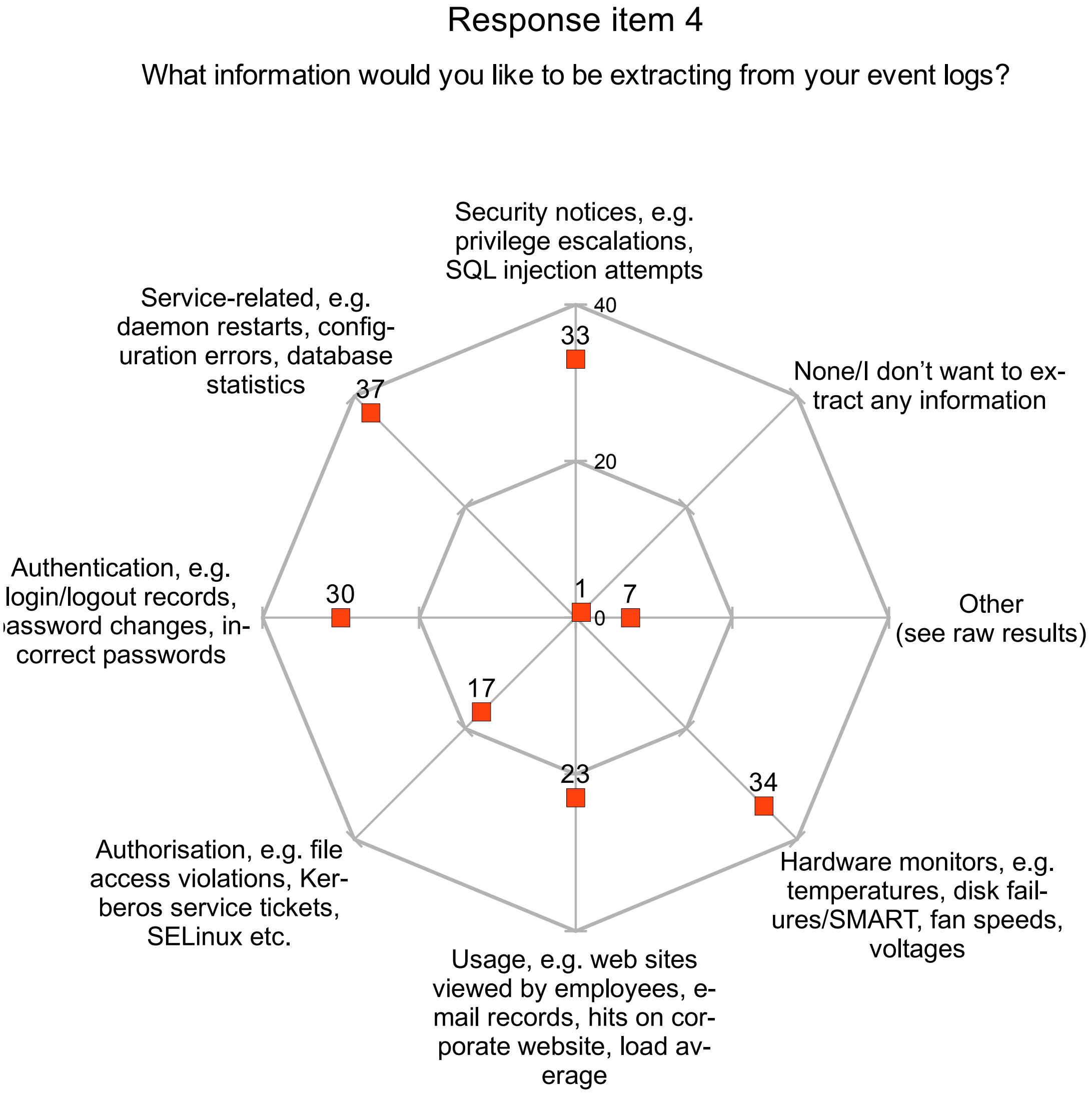


Figure 7.7: Response item 4 results

evidence of an SQL injection means the affected database is fundamentally compromised and will likely have to be either entirely restored from a prior backup or have all recent transactions rolled back to before the time of the injection. The service itself, however, will most likely remain available until sysadmin intervention occurs.

The notion of benign system state is perhaps most lucidly conveyed by the "usage" item (23) : the primary metric here is "what are people doing with the system", which can include factors as diverse as thread/fork load, UNIX load averages, memory and disk consumption, distributed-denials-of-service, percentage of spam e-mail vs legitimate, or even whether "Bob in accounting" is viewing unsavoury content with organisational resources. Rather abstractly, usage can have an effect on service availability, and it is here that sysadmins may need to obtain statistics about performance and resource consumption, for example, in order to forecast new hardware purchases or additional budget for cloud computing. Usage as such is probably the "least bad" of the items presented but was important enough, at least to organisations, for slightly less than half of the participants to select it. It is also emblematic of the densely-packed information space of event log messages (see Section 3.4.1 on page 71) in that usage can be interpreted from so many different possible angles in "normal" event logs with a high value for Weaver's H, while a more uniform log such as one that is application-specific (e.g. Munin) will have a much lower value for H and be much more straightforward to judge for usage purposes.

"Authorisation" is evidently (with only 17 responses) a less serious response item. It must be noted that authorisation is the mechanism which regulates what an *authenticated* identity can or cannot do. This authorisation often takes the form of a permissions mechanism (such as file and directory permissions in a filesystem) that discriminates on a basis of identity – which requires authentication first to verify that identify.

The primary conceptual gap between authorisation and "authentication" is authorisation's lack of a challenge-response motif. It is just a check against a pre-defined list of allowed/disallowed actions. For example, an authenticated user may try to open a file that they are not allowed to, such as the password hash file on a UNIX or UNIX-like system; `/etc/shadow`. This action will fail, and will always fail, unless permission is manually granted by the superuser. Authorisation therefore cannot be teased, gamed, experimented with, or brute-forced. It is either buggy and broken, or not - there is no concept here of authentication mechanisms' "strength" or "weakness". Similarly, it is also much more finely-grained (e.g. per-file permissions vs per-system authentication). Authorisation violations indicate that 'bad behaviour' has been caught and entirely prevented; yet the bad behaviour, and therefore motive, still exists. System integrity is maintained when authorisation is violated and the sysadmin can be notified of *possible/probable intent.* [95]

From the above analysis, it should be possible to synthesize a list of motivations that would drive almost all sysadmins to monitor their systems more closely if they could. The reader should not necessarily be surprised at the position of the last motivation in Table 7.1: systems administrators cannot ever assume the Internet is a 'safe' place.

1	Service availability
2	System integrity
3	Resource consumption levels
4	Intent of attack or compromise

Table 7.1: Motivations in evidence among systems administrators

7.5.3 Indicators of salience

With 41 responses, the source of the message leads the pack of salience indicators for systems administrators in Figure 7.8 on the next page. This finding lends support to the 'impact' dimension and related weighting mechanism presented earlier, in that sysadmins discriminate (perhaps most heavily) by source, independent of the message content. Current systems have no method for including the importance of the source into the log messages issued by that source.

"Textual content" and "correlation" were the same, at 37 responses. While automating the analysis of correlation remains a hard problem [10], and higher quality and more consistent log files have been called for to help with correlation (as in Buckley [26]), another interesting distinction exists between the responses for textual content and numeric content (16). The difference between the response levels (i.e. 37 vs 16) shows the weighting assigned to the two types of log message content by sysadmins; yet no such difference could be identified in the literature on automated log file analysis. Data mining algorithms, neural networks, and static-matching filters of course regard the content as binary streams to either match or analyse for patterns. They do not incorporate even the evidently-higher regard for textual content vs numeric that humans do, yet they attempt to achieve human-like effectiveness.

One base which appears to be relatively frequently utilised by automated mechanisms, perhaps due to its statistical obviousness, is the "number of occurences of a particular message" (29). The Logwatch software project [19] uses this as one of its primary metrics; displaying to the administrator the daily counts of matches for each static pattern-matching regular expression it uses (such as logins and logouts recorded in log files and then later being matched and counted). Outside of well-defined/unchanging and common cases, however, the usefulness of this approach is undermined by trivial differences (to a

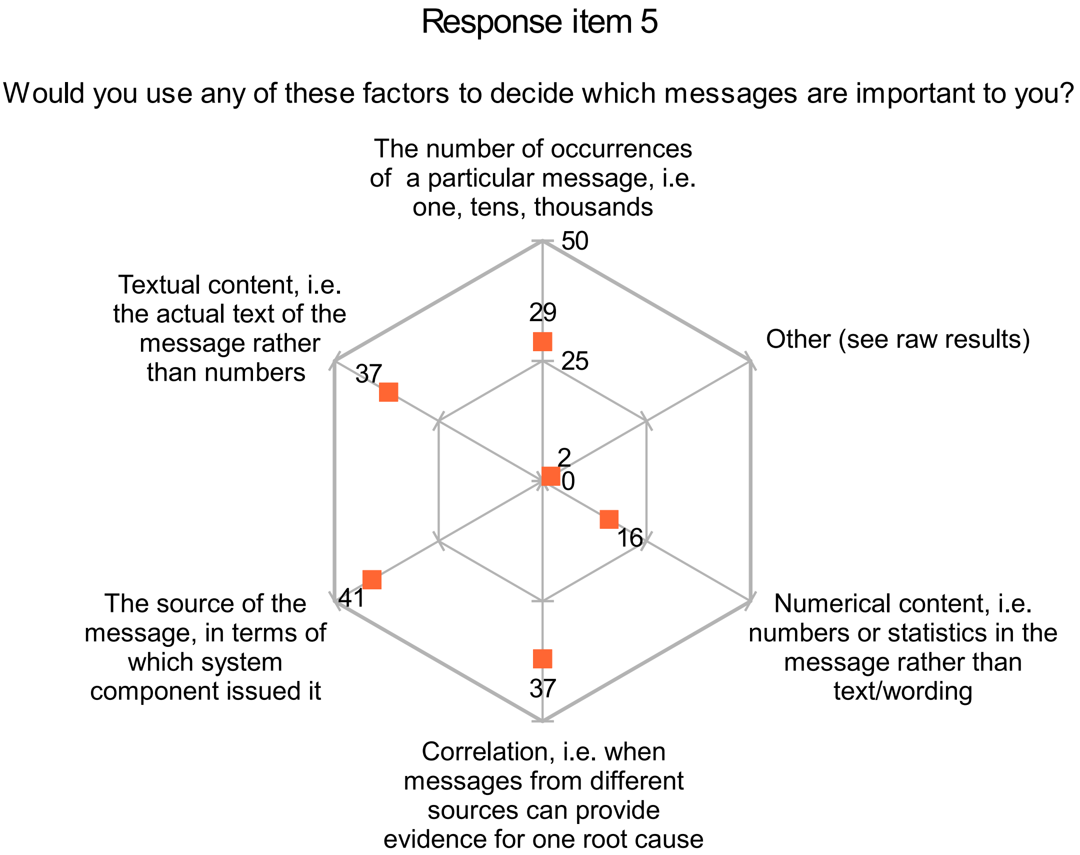


Figure 7.8: Response item 5 results

human observer) that nonetheless greatly degrade any automated mechanism. Atwood [56] pointed out this difficulty with the tools produced by the computer science discipline, in relation to genetic pattern databases (as explained on page 46).

The log file mechanism itself appears highly salient, as can be seen in Figure 7.9; not one participant had gone without the use of a log file in the past year. The overall relevance of event log messages is therefore confirmed for a professional, technical audience.

Asking the audience to rank ten different log messages in terms of their importance raised an issue which has already been alluded to (the likelihood of participants reversing the 1-10 scale). Figure 7.10 on page 122 displays the raw rankings as a stacked bar graph - items with usually-long bar sections were ranked less important and items with shorter bar sections were ranked more important. 13 participants did not provide any rankings while four provided incomplete rankings and two provided rankings with duplicated ranks (e.g. two “9”’s). Figure 7.10 contains this raw data, and the (absent) rankings of the duplicated form 28.

In order to maintain as many data points as possible (necessary with such a small sample), duplicate and missing ranks were altered while maintaining the overall mean. Cleaning

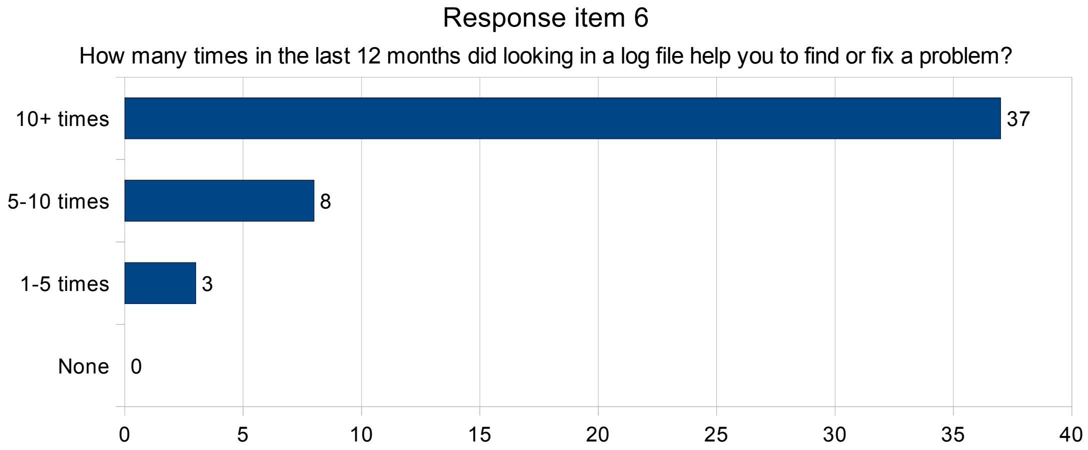


Figure 7.9: Response item 6 results

the data entailed these actions:

- deleting blank rankings
- duplicate ranks were altered to maintain a per-participant mean rank of 5.5
- missing ranks were filled in to maintain a per-participant mean rank of 5.5

The altered forms are detailed in Table 7.2, with the actual alterations in bold italics. These alterations maintain the per-participant ranking mean of 5.5.

Examining the per-item mean and median rankings after these alterations produces the graph in Figure 7.11. Surprisingly (for the author at least) the "Name daemon: clients-per-query..." item was ranked the least serious, with both its mean and median rankings significantly above the standard deviation. Correspondingly, and as expected, the "Kernel: Buffer I/O error on device..." item was ranked as significantly the most important, again outside the standard deviation. It therefore follows that these two items can be considered the 'endpoints', or the extremes of importance, of the items available to be ranked.

Stage two of the cleaning process will eliminate the responses where both the endpoints appear to have been reversed; i.e. where the "Name daemon: clients-per-query..." item was ranked as "1" (most important) *and* where the "Kernel: Buffer I/O error on device..." item was ranked as "10" (least important). This is not intended to ensure a 'perfect' data set where the only responses are consistent with the author's expectations, but rather to retain the largest sample while eliminating the very-likely outliers and enhancing the mean values for each ranking. Reversed rankings act as 'noise', obscuring the true picture. The decision to perform this cleaning was taken as a result of advice from Victoria University of Wellington's consulting statistician.

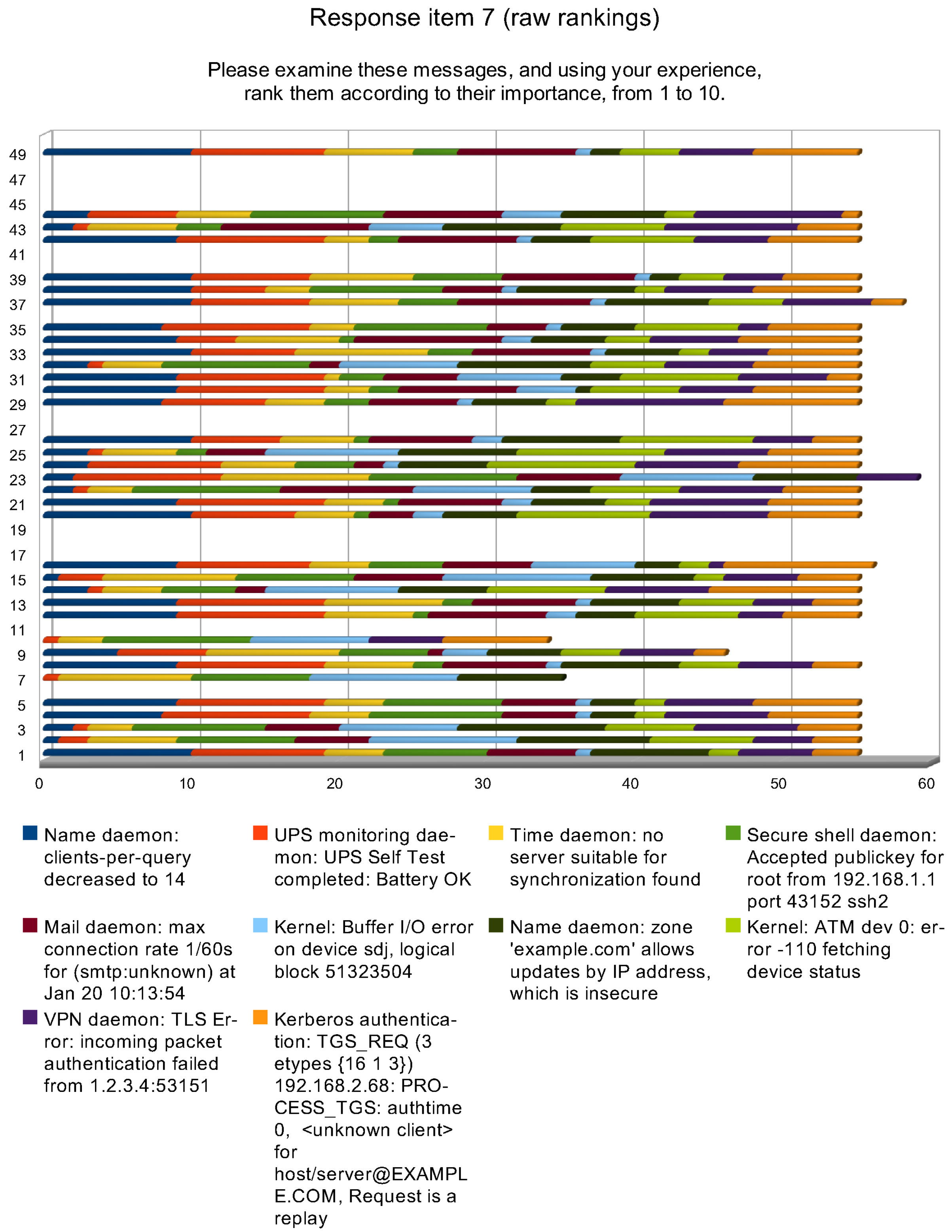


Figure 7.10: Response item 7 results (raw rankings)

Response item ↓ Form→	7	9	10	16	23	37
Name daemon: clients-per-query decreased to 14	***4***	***7.2***	***5.25***	***8.5***	2	10
Time daemon: no server suitable for synchronization found	9	9	3	4	***5.2***	***4.5***
Secure shell daemon: Accepted publickey for root from 192.168.1.1 port 43152 ssh2	8	***7.2***	10	5	***5.2***	4
Mail daemon: max connection rate 1/60s for (smtp:unknown) at Jan 20 10:13:54	***4***	1	***5.25***	6	7	9
Kernel: Buffer I/O error on device sdj, logical block 51323504	10	3	8	7	9	1
UPS monitoring daemon: UPS Self Test completed: Battery OK	1	***7.2***	1	***8.5***	***5.2***	8
Name daemon: zone 'example.com' allows updates by IP address, which is insecure	7	***7.2***	***5.25***	3	7	7
Kernel: ATM dev 0: error -110 fetching device status	***4***	4	***5.25***	2	***5.2***	5
VPN daemon: TLS Error: incoming packet authentication failed from 1.2.3.4:53151	***4***	***7.2***	5	1	4	***4.5***
Kerberos authentication: TGS_REQ (3 etypes {16 1 3}) 192.168.2.68: PROCESS_TGS...	***4***	2	7	10	***5.2***	2

Table 7.2: Statistical corrections to participant responses (first stage of data cleaning)

Response item ↓ Form→	2	15
Name daemon: clients-per-query decreased to 14	1	1
Time daemon: no server suitable for synchronization found	6	9
Secure shell daemon: Accepted publickey for root from 192.168.1.1 port 43152 ssh2	8	8
Mail daemon: max connection rate 1/60s for (smtp:unknown) at Jan 20 10:13:54	5	6
Kernel: Buffer I/O error on device sdj, logical block 51323504	10	10
UPS monitoring daemon: UPS Self Test completed: Battery OK	2	3
Name daemon: zone ’example.com’ allows updates by IP address, which is insecure	9	7
Kernel: ATM dev 0: error -110 fetching device status	7	2
VPN daemon: TLS Error: incoming packet authentication failed from 1.2.3.4:53151	4	5
Kerberos authentication: TGS_REQ (3 etypes {16 1 3}) 192.168.2.68: PROCESS_TGS...	3	4

Table 7.3: Responses deleted due to endpoint reversal (second stage of data cleaning)

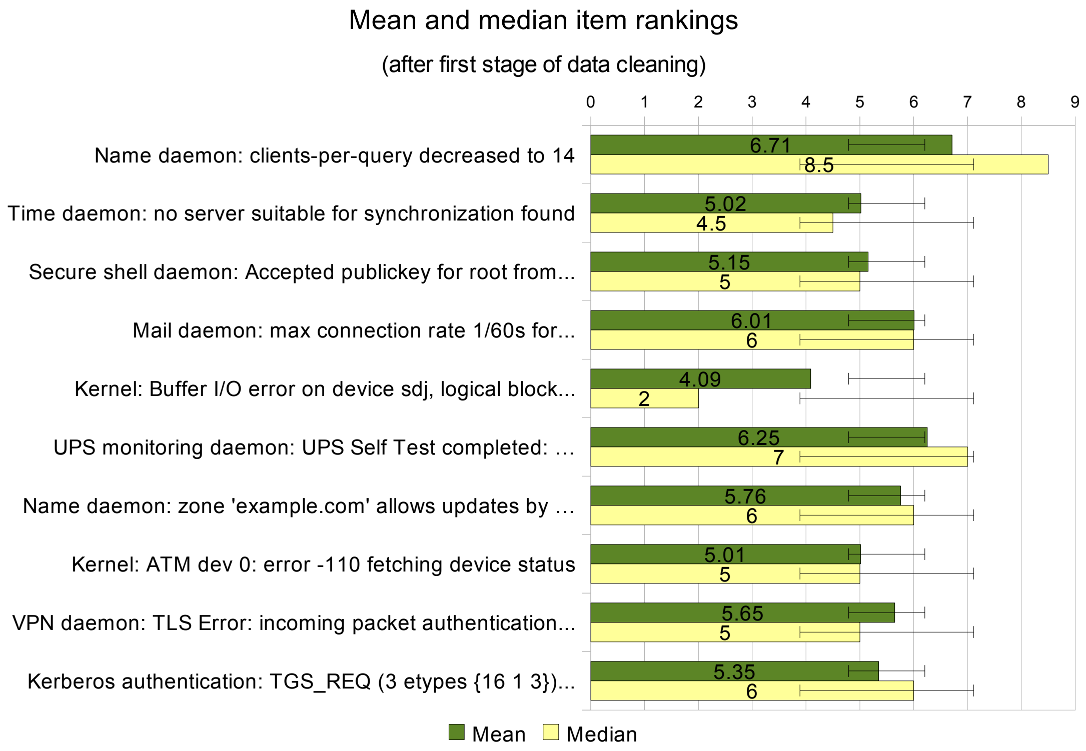
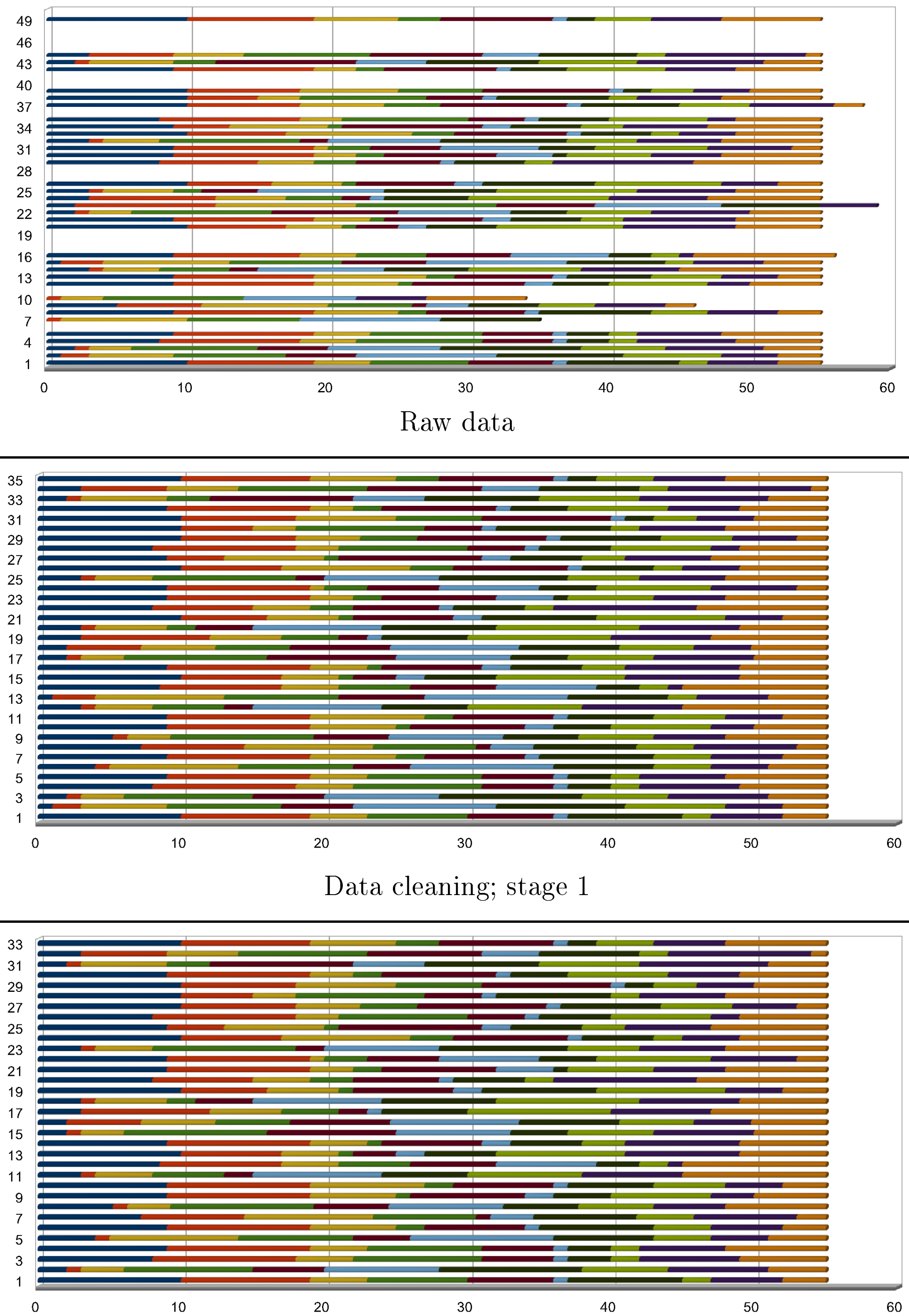


Figure 7.11: Mean and median item rankings after first stage of data cleaning (std. dev. displayed)

The two responses which fitted the end-point criteria (i.e. opposite values given for the response items whose means and medians were outside of one standard deviation) were forms 2 and 15. Those responses are shown in Table 7.3. It is expected that the participants responsible for forms 2 and 15 simply inverted the “1-10” importance scale, which is entirely understandable; as noted earlier, the endpoints were not detailed or ennumerated on the response form. Further surveys could test the acceptance of scales in each direction, with clear instructions about which endpoint is which; or other scale characteristics such as the presence of midpoints and length (number of scale positions).

For comparison, the raw data at each stage is presented in Figure 7.12 on the following page. Note that the vertical axis shows the form numbers independently for each graph, and these numbers cannot be compared across graphs.

A ranking of the importance of the events can be obtained, as shown in Figure 7.13; which is sorted by mean due to there being several identical median values. To validate the list of motivations in Table 7.1 on page 119, the event items will be categorised according to motivation. Table 7.4 shows the outcome of the categorisation. With this small number of event log messages, no conclusive outcome can be drawn; but the distribution of event items still appears to be generally in-line with the order of the motivations. In other words, event log messages which portray service availability or system integrity is-



Raw data

Data cleaning; stage 1

Data cleaning; stage 2

Figure 7.12: Comparison of raw data across stages of cleaning (response item 7)

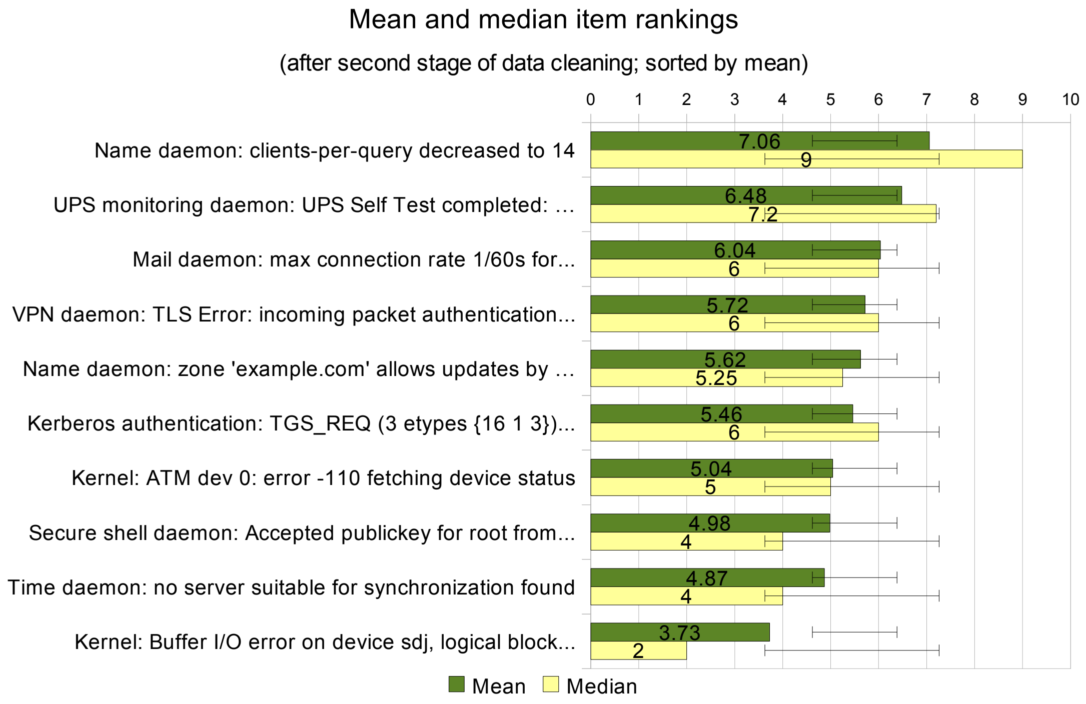


Figure 7.13: Mean and median item rankings after second stage of data cleaning (std. dev. displayed, sorted by mean)

sues are considered more important, while resource consumption and intent of attack are considered less important.

In addition, the distribution of importance may warrant further investigation. To the extent that it has been limited by the small sample size presented here, a larger investigation would provide evidence for a more accurate plotting of systems administrators' concerns. As it is, a curve has been plotted in Figure 7.15, which shows a sharp increase in importance assigned to a small proportion of messages, and a mirrored decrease in importance towards the other end of the scale. Hypothetically, this curve may represent a cognitive model of perception that could be compared to the existing Pareto-like distribution of event log messages.

An ideal event log message handling solution might indeed transform the existing distribution into the identified curve; eliminating the long-tail problem and instead presenting messages on a basis of overall importance or salience, as perceived by the sysadmin, and this hypothesis would need to be investigated. The mechanism of weights applied to the impact metric (discussed earlier) is an initial attempt to realise this goal.

The final response item gathered feedback from the audience as to their disposition towards the proposed "severity, impact, certainty" scale. Figure 7.14 depicts a positive disposition with a mean of "4.89" and median of "5" where "4" is the midpoint.. We must bear in mind that this is an opinion solicited from end-users of the proposed scale and not

Event log message	Motivation category
Name daemon: clients-per-query decreased to 14	Resource consumption levels
UPS monitoring daemon: UPS Self Test completed: ...	Service availability (but with no impact)
Mail daemon: max connection rate 1/60s for...	Resource consumption levels/Intent of attack or compromise
VPN daemon: TLS Error: Incoming packet authentication...	Intent of attack or compromise
Name daemon: zone 'example.com' allows updates by...	System integrity (but with no impact)
Kerberos authentication: TGS_REQ (3 etypes {16 1 3})...	System integrity/intent of attack or compromise
Kernel: ATM dev 0: error -110 fetching device status	Service availability
Secure shell daemon: Accepted publickey for root from...	System integrity
Time daemon: no server suitable for synchronization found	Service availability/system integrity
Kernel: Buffer I/O error on device sdj, logical block...	Service availability (likely impact)

Table 7.4: Categorisation of event log message per-motivation (sorted by mean ranking)

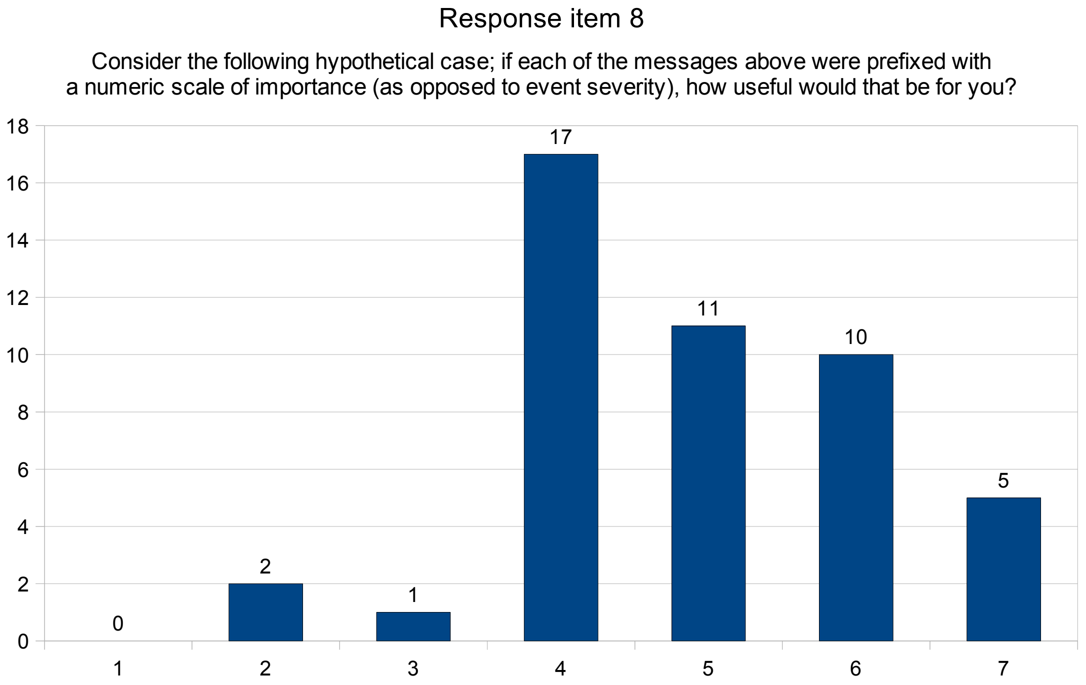
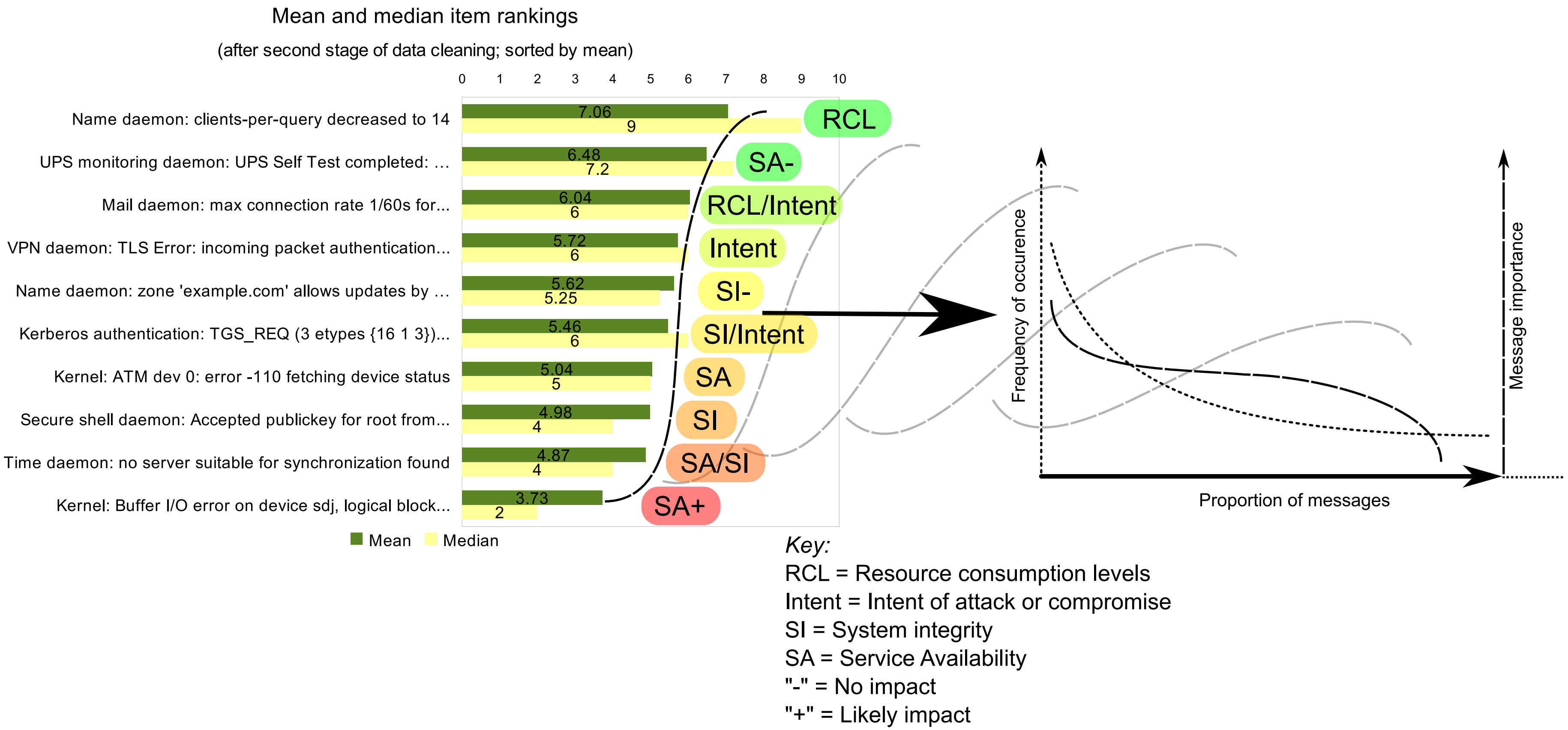


Figure 7.14: Response item 8 results



Key:
RCL = Resource consumption levels
Intent = Intent of attack or compromise
SI = System integrity
SA = Service Availability
"-" = No impact
"+" = Likely impact

Figure 7.15: Hypothetical comparison of 'message importance curve' and Pareto-like event log message distribution

the program developers who would be responsible for implementing the scale as a prefix on the log messages in their source code. It does perhaps indicate a willingness on the part of the audience to improve on the status-quo.

7.6 Summary

The survey, as conducted, has provided what is most likely the first evidence of systems administrators' preferences for event log message processing. Motivations were deduced from the data; these are 'service availability', 'system integrity', 'resource consumption levels' and 'intent of attack/compromise', with SA being most important. While sysadmins appear largely content with the status quo, they also perceive that the situation could be improved. Certainly any improvement could have a significant impact, given that all sysadmins surveyed made (usually extensive) use of their log files.

Chapter 8

Discussion and conclusion

This thesis began with the following premise: event log messages are growing in quantity but the tools available for processing them have not seen anything like an equivalent increase in capability. Regular expressions may have added the occasional improvement in syntax, however, they remain largely identical to their 1980s cousins. Data mining and other statistical approachs (neural networks included) are still probabilistic at best - and only show true promise when painstakingly tuned over long periods of time against a hermetically-sealed data set kept in stasis. The only constant is change, though, and such algorithms simply do not cope with change; often retreating to a position of “new must be bad”. A lack of useful metadata only compounds the situation.

In proposing a metric to improve the event logging landscape, it is worthwhile recalling these motivations and their accompanying responses:

- A data set containing poorly-defined textual descriptions, prone to change, and very densely-packed ‘information’ content (high ratios for Weaver’s H).

    ➠ Provide an embedded, machine-and-human-readable scale. This avoids the need to reverse-engineer human impressions of meaning into the data set - a need created by the discarding of context and background knowledge that is a natural consequence of the event log messaging paradigm.

- Include a notion of context in the larger computing environment.

    ➠ The scale should include a programmer-assigned seed value for the potential *impact* of the event mentioned in the message. This value is then manipulated according to the systems administrator’s preferences and experience in their particular environment.

- Improve the iterative process by drawing on observations of another sector faced with a not-dissimilar problem.

  ➠ Allow systems administrators to rapidly-iterate their impact weights/manipulations on a per-service, per-server/cluster basis. This metric would be exposed to log-analysis tools as a simple scale prefix on existing message text. Other disciplines have recognised the value of splitting scales into multiple dimensions to distinguish between influences on the output value; sysadmins can be better informed if they additionally perceive the programmer's view of severity and certainty for each particular message.

The argument for the scale values being defined in code was developed in [62] and is based on the logic that the programmer or program designer is the person best-acquainted with the problem domain surrounding the issuing of an event log message. 'Severity' as a value in code is already facilitated by RFC 5424 [11] but with less-than-complete uptake since the value is, by default, discarded by Syslog daemons after routing each message. The scope of 'severity', however, is limited to each individual message and simply conveys "how bad" it is on a scale of "0" (worst/most consequential) to "7" (least worst/most inconsequential), as shown in Table 2.1 on page 32.

Unfortunately, such a unidimensional scale cannot hope to accurately convey information about either the consequences for the larger computing environment, or the developer's certainty that their coded values will actually apply in every situation. Even the textual messages themselves do not include this information and it must be added by a human (as was done in Listings 6.4a, 6.4b and 6.4c, beginning on page 156). Person-to-person communication has been enhanced with multi-dimensional scales before; specifically, medicine's Glascow-Coma Scale (GCS) [49] which has been tested and accepted globally as a method for quickly and unambiguously conveying patient triage priority and likely outcome, but there are reservations about summing the dimensions into a single number (e.g. [12]).

The GCS was adopted, in part, to improve information retention when patients were being brought into hospital emergency rooms and then possibly being seen by many different individual medical professionals. Information was prone to being lost; one could characterise this as being like the childhood game of 'Chinese Whispers'. Event log messaging faces a similar challenge caused by the loss of context and background knowledge, not to mention information-overload thanks to 'noise', as information is passed from the mind of the program developer to the mind of the end-user (the systems administrator, in this case). Shannon and Weaver's information model can accurately portray the lossy process and is discussed from page 57 onwards.

8.1 Metric considerations

Any proposed metric for event log messages should aim to facilitate improved information retention by mitigating the effects of filtering (i.e. loss of context, false negatives and positives), noise (false negatives and positives) and volume (cognitive overload). Maintenance demands should also be minimal; continual change can lead to disenchantment with continual maintenance.

As discussed in Chapter 6 on page 97, the metric is designed to improve the lot of the systems administrator. By creating metric seed values in code, the salience-encoding work is performed once (globally) as opposed to millions of sysadmins needing to maintain filters (e.g. lists of regular expressions) which are painstakingly matched to their own needs. Machine-readable metadata reduces the filtering task to a near-trivial exercise.

The metric as presented in this thesis takes the form of a three-dimensional scale, featuring the following dimensions:

- **Severity**: an existing concept drawn from RFC 5424 [11]. Some existing projects, such as OpenVPN [13], have implemented 'severity' to the extent where it is offered as a throttling option for the daemon's logging output; OpenVPN can be configured to discard all log message output which has been seeded with a 'severity' value above this configuration integer (remembering that "0" is the *most important* endpoint). Many projects, perhaps the majority, do not offer any similar capability and their severity values, where they exist, should therefore be treated with a degree of skepticism as they may have been inserted simply to satisfy the logging programming interface. Developers are aware that severity values are simply discarded by default Syslog server configurations.
  The domain scope of 'severity' is limited to the program or daemon issuing the message and as a result the extremes of the scale are calibrated according to the best and worst possibilities for that one entity.

- **Impact**: a synthesis of ideas from X.733 (perceived severity) [43], Wallin et al. (priority correlating poorly with severity) [10], Mitchell et al. (stakeholder salience) [1] and semiotics (the social level above that of mere signs) [6, 54, 3, 61], among others. It refers to the consequences for the larger computing environment of a given event and how these consequences affect humans' real-world priorities. Some events can be very severe in themselves but of minimal impact (e.g. a failed disk in a redundant array) while others can be minimally severe but potentially of great impact (e.g. individual portions of a distributed-denial-of-service attack).

The domain scope of 'impact' is "the forest rather than the trees"; it refers to service provision and availability/organisational-viability overall. The endpoints are calibrated according to the best and worst possibilities for the organisation as a whole.

- **Certainty**: drawn from sources as diverse as Mitchell et al. (urgency/confidence) [1], Matheus et al. (certainty of information reported in battlefield scenarios) [9], Reason (knowledge of human failings with active and passive errors) [60] and Molloy et al. (acknowledgement of the consequences of automation on humans) [58]. 'Certainty', as a scale, is an explicit indication of *doubt*, which exists no matter how much we might crave absolute certainty [79]. Each and every log message is subject to pre-conceived notions when it is coded and depending on the quality of the input data to the preceding conditional statements, and the associated assumptions, the information contained therein may *or may not* be accurate in an as-yet-unknown real world event.
  The domain scope of 'certainty' is as small each individual log message, or perhaps as large as each code block under a given conditional which is acknowledged as relying on less-than-perfect assumptions. For example, a large switch-case structure makes each of its cases, no matter how involved they are, subject to the validity of the initial switch conditional.

The 0-7 scale utilised here is based on the volume of pre-existing software already featuring valid 'severity' values (as documented in RFCs 5424 and 3164), and a desire to facilitate re-use wherever possible. The 0-7 scale does not feature a midpoint as Likert-type scales do (a scale commonly used for opinion surveys in populations, as explained on page 99); it is not a classical "response scale" where the results are expected to form a standard distribution, as no such distribution is either anticipated or in evidence with event log messages.

8.2 Applying weights to the metric

The incorporation of the wider scope associated with 'impact' naturally requires a mechanism which can be customised for each organisation (see Chapter 4 on page 75). Here, this mechanism takes the form of a *weighting* which expresses the relative importance of daemons (i.e. according to the services they provide) and also hosts/servers. It is hypothesized that a sufficiently large number of such weightings, honestly assigned, may begin to resemble a standard distribution: where the distribution is such that most services and machines are “average”, a smaller number are “more important than average” and a similarly small number are “less important than average”. Such a distribution would be distinctly different from the Pareto-like distribution of event log messages with its long tail of rarely-encountered issuings.

Weights readily-configurable by the systems administrator also present an opportunity for rapidly-iterated tuning of the log message output. There are two existing methods for tuning that are both problematic and therefore remain largely theoretical, in that they are hardly ever, if ever, utilised. The first entails submitting a source code patch to the program developer (and acceptance is far from guaranteed) or maintaining one’s own code repository - but obviously these do not apply to closed-source products. The second is the only available avenue for closed-source, although it can also work for FOSS, is a cumbersome string-replacing engine configured with long lists of regular expressions. Such an engine simply recognises pre-configured signatures and either replaces strings or adds information for other tools to make use of later. Weights, in contrast, simply take the form of an integer value in a configuration file, one per service daemon and one for each server’s Syslog forwarding daemon (which scales all impacts across the other daemons according to the importance of that machine).

Note that weights are only proposed here for the ‘impact’ dimension. The much more tightly-scoped ‘severity’ and ‘certainty’ dimensions are intended to enhance communication from developer to end-user on the topics of “how bad” a message was for the issuing entity, and the degree of confidence the developer has in their tests and associated assumptions, respectively. Therefore, at this time, it is difficult to perceive how end-user weights could be valid in those contexts due to the depth of source-level knowledge required to meaningfully influence or second-guess the provided values. Any mechanism for weighting ‘severity’ and ‘certainty’ may also be greatly burdened by their finer granularity: the smaller scopes would entail unique serial numbers on each message-issuing statement in the source code *and* a (potentially very large) lookup table for configuring each and every one.

8.3 Evidence supporting a change to the status-quo

As can be seen in Chapter 7, event logs are used very extensively by systems (including “network”) administrators, to the point where no respondents denied their use. Automated filtering is uncommon, however, with only one third of the audience making any use of such tools. Larger samples from statistics-gathering efforts like the Debian Popularity Contest [21] showing uptake of less than 10% (for FOSS packages installed from distribution repositories). Increasing the awareness of important event log messages is no doubt a worthy goal and one that can be aided by useful, configurable filtering producing timely and relevant results.

The general attitude to existing solutions could perhaps best be characterised as "ambivalent" to "positive". Distinctly few respondents indicated high or very high confidence in their solutions, though, according to Figure 7.5 on page 115. This may indicate awareness of shortcomings or even simply a lack of familiarity with the tools at hand. By far the largest proportion of respondents only use their log files in a reactive manner, leading to knowledge of just how much information is available, but results from other response items (specifically Item 4 on page 117) would imply they then find themselves without sufficient time, motivation or budget to implement a worthwhile mechanism for automated filtering. A quick, effective/straightforward, and free solution is likely to be the proverbial "Holy Grail", even though it violates the similarly-proverbial "fast, good, cheap; pick two" rule.

Any of the existing solutions, however, require a significant investment of time to implement well. An absence of metadata means messages very often contain no indication of their importance - certainly not beyond the scope of their own issuing entity (i.e. 'severity' values, in those cases where they have been responsibly implemented in code: such as in OpenVPN). This means the solution provider has to have interpreted the messages for meaning and importance. Naturally, this interpretation may not match up with the priorities and organisational realities of the systems administrator, yet a statistical or regex-based approach does not allow for tuning without very deep knowledge of the filtering mechanism itself. Lowering this barrier to entry has not been a primary academic aim other than when Buckley [26] circled around the topic in 1992, and again when writing with Siewiorek [25] in 1995.

Both the academic and technical communities needs to remove the barriers to effective event log message filtering to improve system-level awareness; any progress can be assessed against the percentage of systems administrators utilising the relevant tools "in anger". Survey response item 7.8 on page 120 identifies source, textual content and event correlation as the top-three factors in a sysadmin's mind when deciding the importance of messages; yet current solutions ignore such prioritisation of input. Indeed, the relatively-few efforts that have thus far emerged from the computer science discipline focus on statistical heavy-lifting (e.g. [96, 30, 67, 15, 97, 42, 28, 33, 16, 17, 18, 34]) and black-box neural networks (e.g. [10]). It could be said that we have been making our task more difficult through a desire to apply our favoured tools, trying to find a problem for our solution, when in fact the problem in event log messaging is right at the source: inadequate communication from the message's originator (the mind of the designer/developer) to the end-user, thanks to a lack of metadata reflecting real-world concerns. This is an interface shortcoming and we have effectively been trying to use approaches such as data mining to judge inadequate interface output - reconstructing meaning where none yet exists.

8.4 But how can this be improved?

The fundamental issue facing all those who attempt to usefully filter event log messages is a highly-packed, ever-changing data stream (i.e. low redundancy [8]) full of noisy and incomplete communications. Context is critical and yet difficult, at best, to preserve - this is reflected in the challenges encountered in event correlation (e.g. [29, 91, 72]). Improving the lot of event correlation is admittedly not a central goal here although the SIC metric may help.

What is the first step? Upgrading the various aspects (highly packed, cryptic, incomplete, etc) starts with the provision of human- and machine-readable metadata to remove the need to reverse-engineer meaning into messages that can be as short as two characters. This not only relieves a human systems administrator of the research burden involved in finding out whether or not she needs to be concerned about a given message, but also relieves pressure on the academic community to provide a flawless data mining algorithm or even strong AI. The use of a three-dimensional metric is consistent with well-established precedents like medicine's Glasgow Coma Scale.

The second step comes through an incorporation of organisational realities. As implemented in this thesis, these realities are represented with *weights* which, due to reasons of scope compatibility examined earlier, are applied to the 'impact' dimension. The mechanism comfortably facilitates the relative positioning of message sources as more or less important than others: a method for reflecting the per-source criteria selected by sysadmins in survey response item five (see page 120). It also plays a role in highlighting the types of messages that sysadmins are interested in (from survey response item four) by allowing them to select the daemons which they consider to have potential impact on their organisation. Rapid iteration becomes possible with easily-configured numeric weights, so the output can be customised according to personal and/or organisational preference. The message types themselves were broadly categorised into four motivations in Table 7.1; *service availability*, *system integrity*, *resource consumption levels* and *intent of attack or compromise*, which should be of help to developers seeking to provide useful seed values in their source code.

Thirdly and finally, after metadata has been added to event log messages and weights taken into account, the question of exactly how to best utilise this metric remains. Regular expressions cannot usefully reference the metric's values due to combinatorial complexity.

Listing 8.1 Example stanza of rules for filtering event log messages with the SIC metric

```
if (S<=2 && I<=2) then pageSysAdmin;
if (S<=2 && C<=5) then emailSysAdminImmediately;
if (S>2 && S<=6 && I<=5 && C<=5) then emailSysAdminDaily;
if (S=7 && I<=6 && C<=4) then emailSysAdminWeekly;
if (S<=4 && I>=6 && I<=7 && C<=2) then emailSysAdminMonthly;
```

Data mining/neural networks would likely not produce output of any higher determinism either - the addition of more information to an already highly-packed space simply increases the number of possible cases - and these techniques already produce results that their authors often admit are probabilistic, at best.

We should leave the task of interpreting cryptic textual messages meant for human consumption, to humans. Instead, a metadata metric that is machine-readable can itself be used independently of the message's textual content. This step away from textual pattern-matching and statistical analysis (including the various permutations such as corpus construction, data mining, ontological data fusion, neural networks and so on) is core to realising a low-maintenance, tunable alternative which reduces, on a global scale, needless duplication of effort among users of event log messages.

8.5 Realising the alternative

The effective filtering of event log messages remains the aim of this example; a small stanza of pseudo-code rules is shown in Listing 8.1 as a realistic mockup. These rules would largely or entirely replace the regular expressions (which number more than 1000 in a default installation of Logcheck on Debian Linux v5.0), or the algorithms used in data mining or other largely-academic solutions. Note that the scheduled daily, weekly and monthly e-mails have mutually-exclusive content thanks to the bracketing of their 'severity' and 'impact' conditions. The top two rules are for "emergency" situations and may produce duplicate notifications - this is likely to be acceptable given that multiple modes of contact increase the chances of reaching a systems administrator in a timely manner.

Applying the stanza of rules to the outcome mockup presented in Section 6.3 is an exercise that is useful for gauging the feasibility of an 'importance curve', which itself was hypothesised from the survey results in Chapter 7. The importance curve is a worthwhile goal even if for only one reason: it eliminates the long tail which features prominently in plots of event log messages - the large number of messages that occur rarely but make up such a significant proportion of the data stream due to their high degree of variation. It

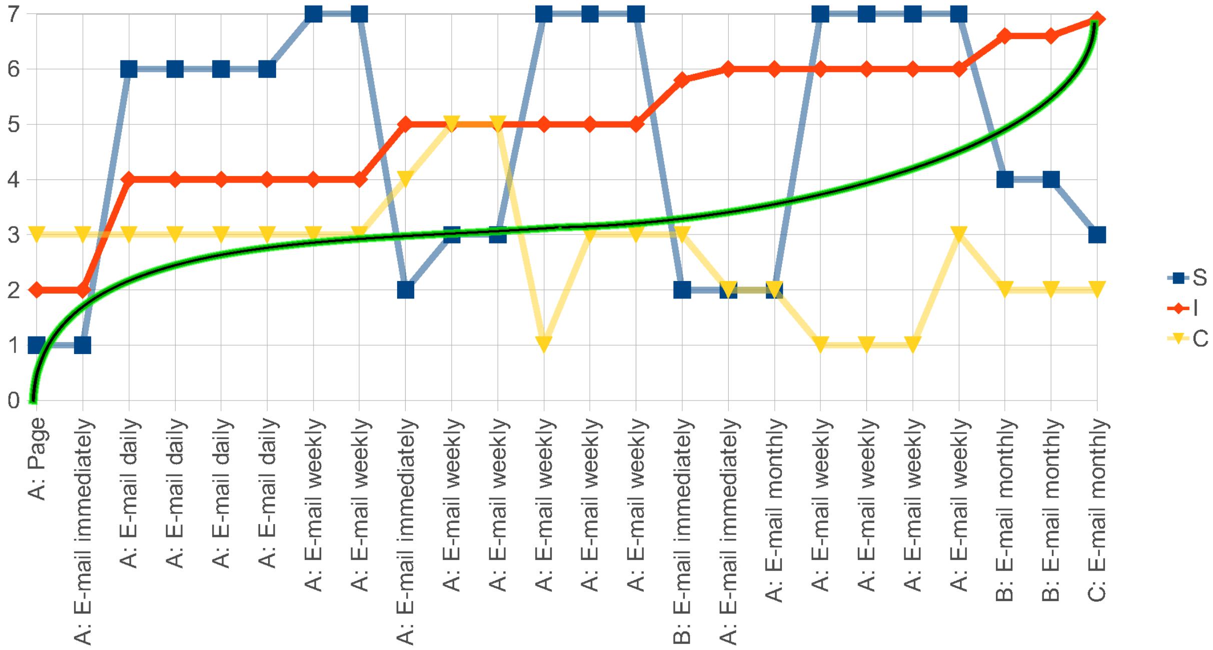


Figure 8.1: Outcome mockup results after filtering with an SIC rule set (with 'importance curve' superimposed)

also represents a possible avenue for further research into human value judgements when faced with such data streams.

The results of the rules' filtering is shown in Figure 8.1. The action to be taken is shown on the X axis, prefixed with the name of the server (A, B or C) which generated the message. The rows of graph data were sorted as they were for the mockup graphs: impact first, severity second, certainty third. It should also be remembered that the rules are only for notification purposes; all messages would still be stored on disk by default, as they are today. Finally, while it might appear anomalous, the first two items ("A: Page" and "A: E-mail immediately") are the same event selected twice by different rules. The message itself is for the motherboard voltage out-of-bounds event. Only the e-mail rules were chosen to produce exclusive output between themselves, as mentioned above; this is due to the identical method and destination of communication.

With the mockup's sample of only 84 messages, no authorititive correlation was expected, and the correlation observed between the 'importance curve' and 'impact' is not strong. Indeed, the mockup sample does not contain any events of "0" severity or impact, but these are highly likely to be observed (albeit rarely) in any large-scale logging environment. One claim that might be made seems at first to be a negative allegation: that values can be intentionally selected via a stanza of rules so that the impact plot line conforms to this, or any other, 'importance curve' - yet this is exactly the capability intended. Systems administrators *should* be able to manipulate their log filtering results to achieve their

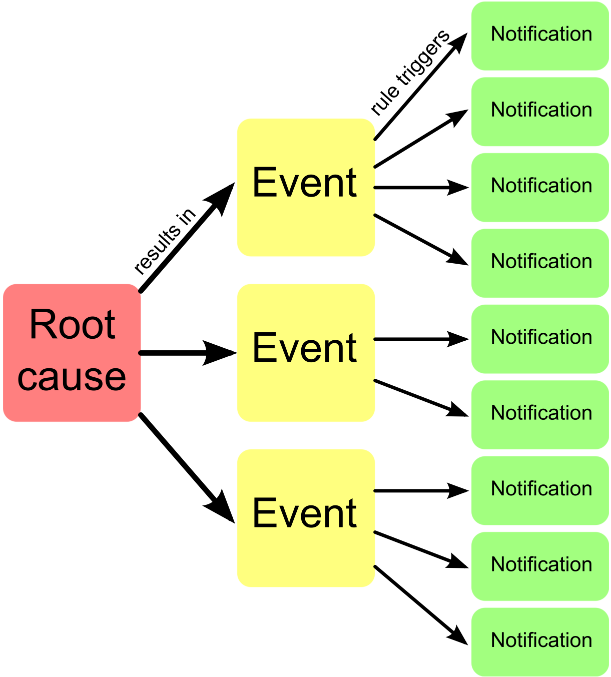


Figure 8.2: One root cause resulting in many notifications

desired outcome but such fine-grained control eludes them with the current toolset. With a sufficient sample of event log messages (e.g. up to one year's worth), and rule tuning, any given 'importance curve' is likely achievable.

One point requiring further work would be whether or not duplicated events should be included in any consideration of an importance curve. This author's initial response was that duplicated events should *not* be included. But what of the duplicated events which share a root cause? For example, it is increasing common for many servers in a datacentre to rely entire on a centralised Storage Area Network (SAN) for disk space, rather than having locally-attached disks in each server's chassis. If the SAN were to fail, or merely respond somewhat slowly to servers' requests for data, a "hailstorm" of event log messages may result from many machines due to the one root cause, as portrayed in Figure 8.2.

Reliably and automatically eliminating such duplicated messages is a hard problem when each server vendor or operating system revision may issue distinct and non-comparable messages about the event. Given these difficult circumstances, is it correct to delete only the duplicated messages caused by overlapping SIC rules, and not others, when considering a plotted line representing importance? To put this question in other words, should an importance curve portray or suppress the notifications which are caused by the use of multiple communication/notification methods? This would be an interesting topic for future research. In any case, duplication of notification is left as a decision for each systems administrator via simple changes to the pseudo-code rules; they could take advantage of the "best-match rule" principle with `break`, `else-if`, or `switch-case` clauses, for example.

The nature of the three-dimensional SIC metric appears to justify itself when focussing on the right-hand half of Figure 8.1. When it comes to selecting a method of communication (i.e. paging, e-mail over different timeframes, etc) in addition to whether or not a message is important in and of itself, the 'severity' and 'certainty' dimensions can be considered in the decision. Thus, server B's message regarding a safe and normal MySQL database shutdown being completed (rated S2, I4, C3) can be selected for immediate e-mailing despite server B not being particularly important. Moreover, the preceding message about beginning the database shutdown (rated S2, I1, C6) is never selected for notification even though it is a "worse" message - simply because the stanza of rules considers the certainty of the message content - with the result of the systems administrator not being needlessly distracted by information about server B which is highly uncertain. This scheme almost permits us a degree of 'atomic operation' when it comes to event log messages: systems that are relatively unimportant can only send more urgent notifications when system state has been established with greater certainty, i.e. *after* a potentially long and involved operation such as a database shutdown has concluded.

Similarly, the very-certain but low impact messages (rated S4, I6, C2) from server B about "SQUAT failed..." are included in a monthly e-mail so these errors are not entirely discarded. SQUAT is an indexing system for the Cyrus IMAP e-mail server software which can improve performance but is by no means necessary for a functioning system. Server C also contributes one message (rated S3, I6, C2) to the notifications: a "404 not found" web server error due to a missing image file. Such an error is highly certain but is an example of an event which alone has minimal impact whereas many (i.e. thousands per day) of these errors may indicate a configuration error or denial-of-service attack, or in the case of tens or hundreds, perhaps a search-engine web spider such as Google or Bing trawling for common files like `robots.txt`. Messages similar in nature to this "404", from system components that are open to malicious attack, may be very certain in relation to each individual message and only gain importance when their rate increases. 'Leaky buckets' or graph analysis could be applied to these quite-severe, minimal-impact, very-certain messages; but this is another topic for future study.

Some readers may query the rate of 'false negatives' which were not selected for notification by the stanza of rules. One of these might be server B's message about a Kerberos authentication failure; the second "[d]ecrypt integrity check failed" one in particular. By itself this message does not indicate any system compromise whatsoever and there may be many hundreds of them daily on any given Kerberos server in a large organisation, all legitimate and due to mistakes when typing passwords. In that sense it is not an error

- no permanent and deterministic failure can be identified, unlike the “404” error above which will always fail when that (likely valid) HTTP request is made. Kerberos errors are extremely likely to be caused by human factors and therefore highly variable in rate, i.e. depending on which individuals are using the system; the “404” is extremely likely to have an automated cause in an HTML page which contains the broken link, or another automated system probing/trawling the web server.

Given these varied scopes, it would be beneficial if the Kerberos server could scale its own message metadata based on event rate - a form of internal and highly-specific ‘leaky bucket’, perhaps. If sufficient research had been performed, the Kerberos server could even incorporate a configuration parameter for the number of people in the organisation and use established statistical data to anticipate a “normal” number of failed authentication attempts within a certain time period: operation outside these bounds would cause the SIC metric to ramp up the ‘impact’ and ‘certainty’ dimensions, resulting in steadily-more-urgent notifications as the situation (such as a password dictionary attack) became more clear.

8.6 Contributions

Here, the objectives presented at the beginning of this thesis will be reviewed and delineated.

1. **Determining the criteria used by systems administrators when assessing system state, including their awareness of the challenges they face with their current automated-filtering toolsets**: Beginning with the concept of a mental understanding informed by intuition, this was refined with the help of survey evidence into a set of four motivations: ‘service availability’, ‘system integrity’, ‘resource consumption levels’, and ‘intent of attack or compromise’. They may not be a definitive list of categories under which all awareness factors can be categorised, but represent a substantial improvement in this area. Evidence was also gathered regarding the uptake of log-filtering technology. Both extensive reasoning and analysis were presented as rationale for introducing architectural modifications to ease difficulties created by the present pace of change.

2. **Distilling the impediments for automated filtering due to communication difficulties and context loss**: Shannon and Weaver’s theory of communication was applied in the context of event log messaging; it exposed latent issues in the communications process between the developer of a program deployed on computing

systems, and the systems administrators responsible for said systems. Semiotic principles were also examined for their contribution to a domain which has seen little, if any, academic consideration of such fundamentals - and helped to strictly define both the modes and methods of communication we currently use, and the nature of the shortcomings. Parallels were identified in other disciplines, and their responses considered as avenues for future improvement in log messaging, in particular; the Glasgow Coma Scale, stakeholder salience theory, military communications, and industrial monitoring.

3. **Based on the impediments identified above, designing representative metadata that can survive a difficult communications process and the loss of surrounding context**: Drawing primarily on two of the established approaches mentioned immediately above (the GCS and stakeholder salience), a three-dimensional metric was proposed: “Severity, Impact, Certainty” (SIC) with each dimension indicated on a separate scale from “0” - meaning the most severe, most profound impact, or least level of certainty - to “7”, meaning the least severe, smallest impact, or highest level of certainty. The GCS provided the template for this metric, given that the GCS was developed for medical triage and prognosis communication purposes in an environment where quick and unambiguous communication is critical. Stakeholder salience, originally meant for identifying the highest priorities for managers in businesses to allocate their time to, was morphed into a systems form, while retaining its useful notions of multiple scopes. Knowledge of each different scope has been utilised to replace the varying dimensional size inherent to the GCS (where each of the three dimensions has a different upper bound).

4. **Including real-world factors that influence decision-making about systems and services**: A simple recognition of differing requirements among individuals and organisations was developed further using semiotics, specifically Stamper’s representational triangle, and can be substantiated with evidence from the survey in Chapter 7. The representational triangle takes information from the human/social level (i.e. roles and importance), represents it using numeric weights, and applies it to the real-world-objects that we use as computing systems (i.e. programs/daemons, servers). The human notion of ‘the bigger picture’ surrounding computing systems, i.e. their context, is thus represented to a certain degree and used to transform the logging output. This is, as far as this author can establish, the first time that a feedback or tuning mechanism has been included in an event-logging system architecture.

5. **Integrating all the steps above into a deterministic filtering alternative**

that offers potential improvements compared to existing approaches: This item refers specifically to the software improvement goals listed in Section 1.4 on page 27.

(a) Reduced duplication of effort.

Achieved. The existing approaches require each site or systems administrator to customise a substantial list (over 1000 items by default) of regular expressions: the burden of interpretation and action is largely on the end-user, rather than the developer. Shifting a very small proportion of this burden back to the developer, through the creation of simple numeric prefixes for each message, removes a much greater proportion of the workload from the collective user base (i.e. systems administrators) by eliminating much or all of the duplicated interpretation, and additionally distilling the method of action into a simple conditional statement.

(b) Deterministic and reproducible output.

Achieved. An architecture based on numeric thresholds and a simple scaling function will produce identical output given identical input. The same cannot be said for statistical approaches such as data mining or neural networks. The architecture presented here evalutes every line separately, leaving the higher-level relationship analysis either to a human or a downstream system, whereas data mining, neural networks, and the like attempt to employ poorly- or undirected relationship analysis at the first stage and are therefore burdened by the message-interpretation problem.

(c) Minimised maintenance requirement in normal operation.

Likely achieved, but this remains to be proven. Systems based on regular expressions require maintenance upon many of the changes that are necessitated by running a computing environment of any substance (security updates or patches, for example). Until such maintenance is carried out, any altered messages are likely to be tagged as possible attacks. Statistical algorithms require time for new events to cross a threshold of experience and/or statistical significance as well as a constant time investment in training to obtain optimal effectiveness. Neural networks combined with trouble-ticket databases (as in Wallin et al. [10]) are an attempt to re-purpose existing data as a training set, but inevitably hit problems due to mismatches in data-set purpose with the result of a “correct” output rate of only 50%.

In comparison with ‘the competition’ above, the SIC architecture has a miniscule amount of configuration to deal with: only the stanza of rules and a single weighting value for each contributing component such as a daemon or server. Interpretation metadata in a machine-readable form frees an event-log-message filter to simply *filter* - resulting in a compact and maintainable stanza of rules for laying out personal or organisational thresholds. The SIC architecture has yet to be deployed in a large-scale environment, though, so cannot be regarded as proven.

(d) Customisable policies that manipulate the output according to personal/organisational preferences and support multiple modes of communication determined by an event's real-world priority.

Achieved. The stanza of rules is a mechanism for enacting such policies. Its capabilities are limited only by those of the language used to code it, e.g. Perl or Python, and the modes of communication which can be scripted. Service Level Agreement (SLA) policies could be enacted as graph tolerances around an established known-good history, such as "operation outside a 5% tolerance shall be considered to be *in fault*", if graph analysis techniques were applied to the three dimensions of the SIC metric. At perhaps the most basic level, means and medians for each dimension could be tracked within windows of one hour, one day, one week, one month, or any other given timeframe, to detect abnormal activity.

(e) Improved communication, with greatly reduced ambiguity, from programmer/program designer to end-user/systems administrator.

Likely achieved. While the SIC metric was designed with input from literature on triage, salience identification and priority management, to deal with the communication deficit eluciated with the help of Shannon, Weaver, and semiotics in general, it has not been formally considered from a psychological point of view. SIC attempts to encode the contextual background knowledge of the programmer/program designer so that a deeper context can be included with each message and in a form which survives the filtering process. It has been largely supported by the results of a small survey, but this is not a rigorous examination.

What is clear is that SIC does provide a machine-readable piece of metadata which can be assessed without the need for background research into the meaning of the message. A systems administrator can therefore save time by filtering out messages with trivial content. The unambiguous nature of numeric scales allows for more effective communication across cultures and languages, reducing or eliminating the reading of source code (the action of which is itself a reversion to a standardised and less-ambiguous form of communication) and other forms of "second-guessing".

8.7 Future work

The research and premises set out in this thesis lead directly and indirectly to some interesting avenues for future work.

- **Deploying the SIC metric in a large-scale trial**. For example: the writing of patches applied to source code to create SIC-enabled daemons which are then deployed at an Internet Service Provider. A significant trial would provide invaluable data with which to test the concept of the 'importance curve', as well as elucidating a possible distribution that server and daemon weights conform to.

- **Further analysis of SIC's dimensions**, particularly validity testing against established psychological/cognitive guidelines. It should be remembered that the dimensions are not currently intended to be *response* scales suitable for gathering data expected to conform to a standard distribution; but rather they should represent, as far as is possible, a linear progression from endpoint to endpoint. From a numeric viewpoint this is of course trivial. The situation becomes more challenging, however, once one seeks to enrich the scale with sets of example events for guiding program developers in assigning their seed values.

- **Optimisation of rule stanzas: is there a valid default setting?** Following on from a large-scale trial, an interesting research topic would be the evaluation of a variety of 'importance curves' produced by their associated rule stanzas. That is, which curve represents the best overall starting point, or default setting, for a SIC-enabled logging system? Or is the curve too closely associated with an individual's personal preference to be separately discerned or indeed generalised across systems administrators? Optimisation should also consider the role of 'leaky buckets', duplicated messages (whether from an identical root cause or rule overlap) and other issues pertinant to dealing with a variety of systems issues.

- **Investigating the effectiveness of a straight-line scaling function**. As shown in Section 6.2 on page 100, the proposed scaling function (for applying the assigned weights to the 'impact' dimension) achieves an entirely linear result, in the name of simplicity. It remains to be seen if this is actually the best outcome: there may well be scenarios where a logarithmic result is preferable, for example. Certain system or daemon roles may be best served by particular scaling functions - this is unknown at the present moment.

- **A deeper analysis and cross-correlation of the survey results**. This author does not have sufficient experience with survey analysis to have produced an 'exhaustively interpreted' set of results. There may be underlying nuances and correlations which have not been identified.

- **Evaluation of SIC beside academic alternatives such as data mining algorithms, on the same data set**. With a large sample of SIC-enabled data, the usefulness of the metric to other academic approaches could be evaluated. Data mining or neural networks may indeed be able to produce improved filtering results when SIC metadata is present; having a common source data set would allow direct comparisons to be drawn.

8.8 Publications

The research conducted for this thesis was the basis for one publication:

- Radford, P., Linton, A., Welch, I. Accepted for publication. Event log messages as a human interface. *Proc. OZCHI Brisbane 2010.*

Chapter 9

Appendix A: Logcheck README

The following README file is located at:

"`/usr/share/doc/logcheck-database/README.logcheck-database.gz`"

following installation of the logcheck-database package (version 1.2.69) on Debian Linux v5.0 [8]. The package is released under the terms of the General Public License (GPL), version 2. As with all software, this information is subject to change.

SYNOPSIS

Logcheck-database provides the egrep patterns required by the package "logcheck"; they are used to filter recent log messages (collected using "logtail") into a mailed news summary.

SETS OF RULES

There are three layers of sets of filtering rules, all of which are normal egrep pattern-matches, applied in turn.

1. the "SECURITY ALERTS" layer, designed to detect the traces of active intrusion attempts. Patterns raising the alarm go in " etc logcheck cracking.d"; any event that matches one of these patterns turns the report into an urgent "Security Alerts" report, with the relevant event moved to a special section. The cracking.d standard keywords file is seeded with known symptoms of hostile activity (see logcheck's README.keywords file). Patterns cancelling such maximum-priority alarms are not used in the default logcheck configuration, but if the local administrator enables this layer of filtering in logcheck.conf, then the rules go in the directory " etc logcheck cracking.ignore.d".

Matches with cracking.ignore rules will then reclassify the alert as a false alarm (compare violations.ignore below). Note that this means they are totally ignored - log messages handled at one layer are not carried over to lower layers.

2. the "SECURITY EVENTS" layer, designed to detect less critical events still considered worthy of special attention. Patterns raising the alarm go in " etc logcheck violations.d"; matches with these result in a "Security Events" alert, with the relevant event moved to a special section. Patterns cancelling such alarms go in the standard directory " etc logcheck violations.ignore.d"; apparent "Security Events" that match with violations.ignore patterns are discarded as false alarms.

3. the "SYSTEM EVENTS" layer, handling leftover log messages. This layer doesn't have an equivalent to the alarm-raising cracking.d and violations.d; instead _all_ remaining lines from the logfiles are considered for inclusion in the main "System Events" section. Patterns in the three " etc logcheck ignore.d.*" directories again function to overrule alerts; the log messages that match them are excluded from the report as trivial. The specific directories consulted depend on the prevailing logcheck "REPORTLEVEL" (for details see the corresponding README for logcheck). The bare minimum is the set of filters in ignore.d.paranoid. When _no_ logged events make it through the filters no report is mailed.

FILES WITHIN EACH DIRECTORY

Each of the rules-directories can contain pattern files of the following kinds:

. <packagename> The rule filename must only contain characters compatible with run-parts(8). As of this writing, this includes alphanumeric characters, underscore, and hyphen. Contains filters relevant to only one Debian package - for example if "fooserver" logs suspicious events like this: "$DATE $HOSTNAME fooserver[$PID]: $USER is up to no good" then a line in " etc logcheck violations.d fooserver" with an appropriate pattern will promote it from a mere "System Event" to a full "Security Event" in a subsection of the mailing headed "fooserver". Or then again if that kind of log message is more trivial than it looks (maybe "foo" is a networked game of spy-and-counterspy) then a line in " etc logcheck- ignore.d.server fooserver" will turn it into a nonevent for all but the most assiduous of administrators. Sometimes a package will have not only special alarm calls which _do_ need to be "Security Events" triggers but also exceptional variants which _don't_ - maybe it logs either “$DATE $HOSTNAME fooserver[$PID]: $USER barred" or “$DATE $HOSTNAME fooserver[$PID]: none barred". In this situation the alarm can be overruled by a violations.ignore rulefile named "fooserver" which filters "none barred". This will _not_ affect other "Security Events" featuring the words "none barred" (that might allow crackers to use those words to cover attacks on ssh). Instead, any <packagename> ignore-files only affect the log messages that would have been in that package-specific report section. Apart from anything else this limitation reduces the number of rules that need to be processed.

. logcheck or . logcheck-<packagename> Standard "generic" rules go in each directory's ". logcheck" file; thus for instance any log message at all matching "ATTACK" (listed in " etc logcheck cracking.d logcheck") _always_ triggers a "Security Alert", unless you deliberately tamper with "cracking.ignore.d" rules. ** Debian Note: we emptied out . logcheck and merged all . logcheck-<packagename> files into the ignore.d.* <packagename> files. This was done because the standard rules in . logcheck matched too many false positives (see e.g. #449028) and resulted in a lot of rule duplication (#254542).

Remember that package-specific "ignore" filters will _not_ override non-package-specific "flagging" patterns! Thus for instance if "fooserver" outputs syslog messages like this: "$DATE $HOSTNAME fooserver[$PID]: 3 attempts 0 rejected" then the standard keyword "reject" listed in the generic " etc logcheck violations.d logcheck" file will trigger frequent "Security Events" reports. Putting a filtering pattern in

" etc logcheck violations.ignore.d fooserver"

won't help here! The solution is to use a file named in the specially-privileged . logcheck-<packagename> format:

" etc logcheck violations.ignore.d logcheck-fooserver".

This can contain patterns provided by that particular package which nonetheless need to take precedence over the generic rules.

. local or . local-<packagename> Sysadmins can use the "local-*" filenames to create their own additions to the "logcheck-*" pattern lists. If you have "ippl" logging network connections verbosely into syslog then you can put custom "Security Events" keywords in
" etc logcheck violations.d local-ippl" and exceptions in
" etc logcheck violations.ignore.d local-ippl".

WRITING RULES

Be careful when editing local rule files; logcheck will preprocess them to eliminate dangerous blanks (since "egrep '' syslog" matches every line) and comment lines, but some attention is needed when composing custom patterns to avoid excessively generous filtering. The objective in logcheck rules is to match precisely the target log messages and no more, using all the resources of Extended Regular Expressions. If you're sick of reading log messages like this:

```
Apr 6 19:30:24 oempc wwwoffled[11763]: WWWOFFLE Online.
```

```
Apr 6 19:31:54 oempc wwwoffled[11763]: WWWOFFLE Offline.
```

...then the local ignore pattern you need is something like this:

```
^\w{3} [ :0-9]{11} oempc wwwoffled\[[0-9]+\]: WWWOFFLE (On|Off)line\.$
```

The characters ".?*+[](){}^$|\" are "special" in extended-regexps, so they need to be escaped if intended literally (like the final stop in the example above). Be especially wary of unbalanced brackets, which can choke egrep. Local administrators can afford to be more specific than the package maintainers who provide filters for "fooserver" etc. You can take the locale for granted, saying "[a-zA-Z]" where package maintainers should be using "[[:alpha:]]"; and you can write out things like hostnames explicitly - hence "oempc" above, rather than the pattern "[._[:alnum:]-]+".

TESTING RULES

To test new rules, you can grep your log file, and remove trailing space with something like this:

```
sed -e 's [[:space:]]*$   ' var log syslog | egrep \ '^\w{3} [ :0-9]{11} oempc wwwoffled\[[0-9]+\]: WWWOFFLE (On|Off)line\.$'
```

If the log line is displayed, then your regex works. Pass all rules files through "sort -u" to simplify maintenance, then ensure they have a final end-of-line carriage return so that they "cat" nicely. Since System Events aren't subdivided by package, it makes no difference whether ignore.d.* local rules are split up into "local-x", "local-y" and "local-z" or merged into one "local" file; use whatever's convenient. Another safety-net is provided by the fact that the process that collates all the applicable rules uses "run-parts", the standard Debian utility also used for iterating through " etc cron.d", " etc ppp ip-up.d" etcetera. It therefore automatically ignores files with names such as "fooserver.disabled" or "local~".

SUBMITTING RULES

If there are messages which are not ignored by logcheck that should be, file a bug against the package logcheck-database in the Debian Bug Tracking System (BTS). If you're new to the reporting bugs using the Debian BTS, you can learn more at: http: www.debian.org Bugs Reporting

Unfortunately, we don't have the time to add and update rules for everything, therefore the following exceptions apply:

- Debug messages
- Messages produced by software not included in Debian
- Temporary messages which are due to a bug in the package
- Messages related to daemon startups and shutdowns

Please do not file bugs related to these messages.

Chapter 10

Appendix B: Outcome mockup

The messages in this appendix are genuine and have been modified to contain prefixes with “S, I, C metric” values. Full-size versions of the per-server graphs on page 105 are also included here. The tuples which were graphed were first sorted by their ‘impact values’, then each impact bracket was sorted by their ‘severity’ values, then each impact+severity bracket was sorted by their ‘certainty’ values. This is effectively a “zooming-in” by scope, where ‘impact’ is the largest scope and ‘certainty’ is the smallest scope.

Listing 10.1 Server A: mocked-up event log messages

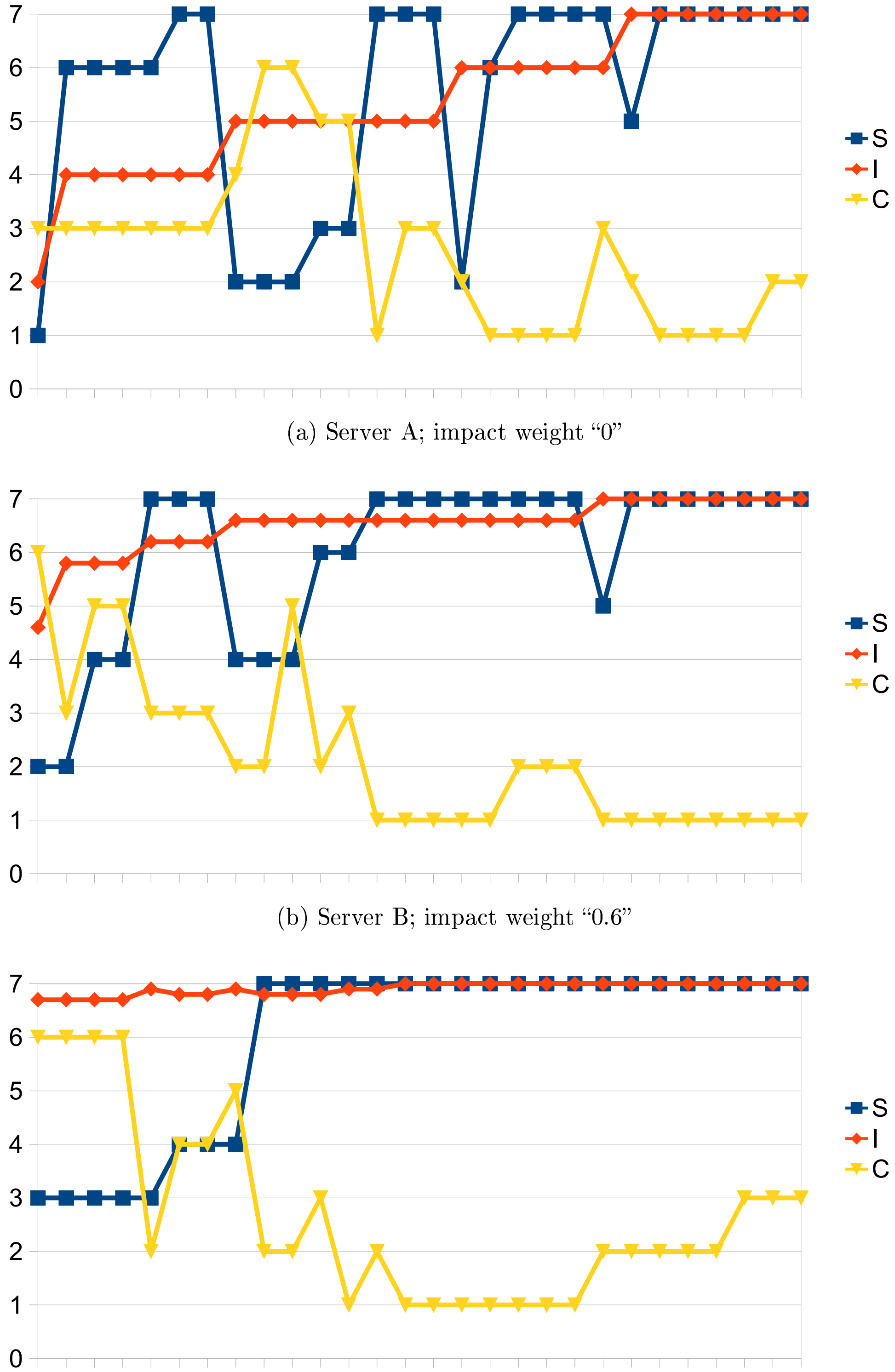
```
Jan 30 06:56:45 serverA postfix/smtpd[31944][S6,I4,C3]: connect from unknown
    [27.57.212.209]
Jan 30 06:56:45 serverA postfix/smtpd[31944][S3,I5,C5]: warning: 27.57.212.209: hostname
    Static-209.212.57.27.airteldataservices.com verification failed: Name or service not
    known
Jan 30 06:56:50 serverA postfix/smtpd[31944][S6,I4,C3]: connect from unknown
    [190.51.227.124]
Jan 30 06:56:50 serverA postfix/smtpd[31944][S6,I4,C3]: disconnect from unknown
    [27.57.212.209]
Jan 30 06:56:50 serverA postfix/smtpd[31944][S2,I5,C6]: lost connection after CONNECT
    from unknown[27.57.212.209]
Jan 30 06:56:50 serverA postfix/smtpd[31944][S3,I5,C5]: warning: 190.51.227.124: hostname
     190-51-227-124.speedy.com.ar verification failed: Name or service not  known
Jan 30 06:56:54 serverA postfix/smtpd[31944][S2,I5,C4]: NOQUEUE: reject: RCPT from
    unknown[190.51.227.124]: 554 5.7.1 Service unavailable; Client host [190.51. 227.124]
     blocked using bl.spamcop.net; Blocked - see http://www.spamcop.net/bl.shtml
    ?190.51.227.124; from=<hiblancheg@yowzahost.com> to=<account@example.com> proto=ESMTP
     helo=<hm5slz4q.yi>
Jan 30 06:56:56 serverA postfix/smtpd[31944][S6,I4,C3]: disconnect from unknown
    [190.51.227.124]
Jan 30 06:56:56 serverA postfix/smtpd[31944][S2,I5,C6]: lost connection after DATA (0
    bytes) from unknown[190.51.227.124]
Jan 30 07:45:58 serverA ntpd[2304][S5,I7,C2]: kernel time sync status change 0001
Jan 30 08:26:02 serverA postfix/smtpd[2539][S7,I4,C3]: connect from bulkmail1.freecycle.
    org[94.102.157.234]
Jan 30 08:26:04 serverA postfix/cleanup[2543][S7,I5,C3]: 0617DB6A5: message-id=<E1PjGQo
    -0001Er-G6@bulkmail1.freecycle.org>
Jan 30 08:26:04 serverA postfix/qmgr[28214][S7,I5,C3]: 0617DB6A5: from=<post
    -12594045-8691005@bounces.freecycle.org>, size=2626, nrcpt=1 (queue active)
Jan 30 08:26:04 serverA postfix/qmgr[28214][S7,I6,C3]: 0617DB6A5: removed
Jan 30 08:26:04 serverA postfix/smtpd[2539][S7,I6,C1]: 0617DB6A5: client=bulkmail1.
    freecycle.org[94.102.157.234]
Jan 30 08:26:05 serverA postfix/smtpd[2539][S7,I4,C3]: disconnect from bulkmail1.
    freecycle.org[94.102.157.234]
Jan 30 08:29:25 serverA postfix/anvil[2541][S7,I7,C1]: statistics: max cache size 1 at
    Jan 30 08:26:02
Jan 30 08:29:25 serverA postfix/anvil[2541][S7,I7,C1]: statistics: max connection count 1
     for (smtp:94.102.157.234) at Jan 30 08:26:02
Jan 30 08:29:25 serverA postfix/anvil[2541][S7,I5,C1]: statistics: max connection rate
    1/60s for (smtp:94.102.157.234) at Jan 30 08:26:02
Jan 30 08:45:01 serverA CRON[3057][S7,I7,C1]: pam_unix(cron:session): session closed for
    user root
Jan 30 08:45:01 serverA CRON[3057][S7,I7,C2]: pam_unix(cron:session): session opened for
    user root by (uid=0)
Jan 30 08:45:01 serverA CRON[3059][S7,I7,C1]: pam_unix(cron:session): session closed for
    user root
Jan 30 08:45:01 serverA CRON[3059][S7,I7,C2]: pam_unix(cron:session): session opened for
    user root by (uid=0)
Jan 30 08:47:21 serverA sensord[S7,I6,C1]:   +12V: +11.94 V (min = +11.38 V, max = +12.57
     V)
Jan 30 08:47:21 serverA sensord[S1,I2,C3]:   +3.3V: +3.12 V (min = +3.13 V, max = +3.45 V
    )
Jan 30 08:47:21 serverA sensord[S7,I6,C1]:   CPU Temp: 34.3 C (limit = 65.3 C, hysteresis
     = 60.2 C)
Jan 30 14:57:30 serverA squid[3123][S2,I6,C2]: Squid Parent: child process 3125 exited
    with status 0
Jan 30 14:59:37 serverA squid[3118][S6,I6,C1]: Squid Parent: child process 3120 started
```

Listing 10.2 Server B: mocked-up event log messages

```
Feb  1 16:04:51 serverB krb5kdc[2656][S7,I6,C2]: AS_REQ (3 etypes {16 1 3}) 172.16.1.100:
      NEEDED_PREAUTH: example@EXAMPLE.COM for krbtgt/EXAMPLE.COM@EXAMPLE.COM, Additional
     pre-authentication required
Feb  1 16:04:51 serverB krb5kdc[2656][S4,I4,C5]: AS_REQ (3 etypes {16 1 3}) 172.16.1.100:
      PREAUTH_FAILED: example@EXAMPLE.COM for krbtgt/EXAMPLE.COM@EXAMPLE.COM, Decrypt
     integrity check failed
Feb  1 16:04:51 serverB krb5kdc[2656][S4,I4,C5]: preauth (timestamp) verify failure:
     Decrypt integrity check failed
Feb  1 17:53:35 serverB spamd[8148][S7,I7,C1]: prefork: child states: II
Feb  1 17:53:35 serverB spamd[8206][S7,I6,C1]: spamd: clean message (-3.5/5.0) for nobody
     :65534 in 1.7 seconds, 8715 bytes.
Feb  1 17:53:35 serverB spamd[8206][S7,I5,C3]: spamd: result: . -3 - AWL,BAYES_00,
     HTML_IMAGE_RATIO_08,HTML_MESSAGE,MIME_HTML_ONLY,MIME_QP_LONG_LINE,RCVD_IN_DNS WL_MED
     scantime=1.7,size=8715,user=nobody,uid=65534,required_score=5.0,rhost=localhost,raddr
     =127.0.0.1,rport=56269,mid=<31f101$a0vuv3@tmmta2.akl.trad eme.local>,bayes=0.000000,
     autolearn=ham
Feb  1 18:15:16 serverB cyrus/imap[28222][S7,I7,C1]: executed
Feb  1 18:15:16 serverB cyrus/imapd[28222][S7,I5,C3]: accepted connection
Feb  1 18:15:16 serverB cyrus/master[28222][S7,I7,C1]: about to exec /usr/lib/cyrus/bin/
     imapd
Feb  1 18:15:18 serverB cyrus/imapd[28222][S4,I6,C5]: TLS engine: No CA file specified.
     Client side certs may not work
Feb  1 18:15:18 serverB cyrus/imapd[28222][S6,I6,C3]: starttls: TLSv1 with cipher AES256-
     SHA (256/256 bits new) no authentication
Feb  1 18:15:19 serverB cyrus/imapd[28222][S7,I5,C3]: login: station.example.com
     [172.16.2.3] example plain+TLS User logged in
Feb  1 18:15:20 serverB cyrus/imapd[28222][S7,I7,C1]: open: user example opened INBOX
Feb  1 18:15:20 serverB cyrus/imapd[28222][S7,I7,C1]: seen_db: user example opened /var/
     lib/cyrus/user/e/example.seen
Feb  1 18:25:13 serverB cyrus/imapd[28472][S4,I6,C2]: SQUAT failed
Feb  1 18:25:13 serverB cyrus/imapd[28472][S4,I6,C2]: SQUAT failed to open index file
Feb  1 19:10:15 serverB krb5kdc[2656][S7,I6,C2]: AS_REQ (3 etypes {16 1 3}) 172.16.1.100:
      ISSUE: authtime 1296540615, etypes {rep=16 tkt=16 ses=16}, example@EXAMPLE.COM for
     krbtgt/EXAMPLE.COM@EXAMPLE.COM
Feb  1 19:10:15 serverB krb5kdc[2656][S6,I6,C2]: AS_REQ (3 etypes {16 1 3}) 172.16.1.100:
      NEEDED_PREAUTH: example@EXAMPLE.COM for krbtgt/EXAMPLE.COM@EXAMPLE.COM, Additional
     pre-authentication required
Feb  1 19:10:15 serverB krb5kdc[2656][S7,I6,C2]: TGS_REQ (3 etypes {16 1 3})
     172.16.1.100: ISSUE: authtime 1296540615, etypes {rep=16 tkt=16 ses=16},
     example@EXAMPLE.COM for host/serverB.example.com@EXAMPLE.COM
Feb  1 19:25:19 serverB cyrus/ctl_cyrusdb[31955][S7,I6,C1]: archiving database file: /var
     /lib/cyrus/mailboxes.db
Feb  1 19:25:19 serverB cyrus/ctl_cyrusdb[31955][S7,I6,C1]: archiving log file: /var/lib/
     cyrus/db/log.0000000011
Feb  1 19:25:19 serverB cyrus/ctl_cyrusdb[31955][S7,I6,C1]: checkpointing cyrus databases
Feb  1 19:25:19 serverB cyrus/ctl_cyrusdb[31955][S7,I7,C1]: done checkpointing cyrus
     databases
Feb  1 19:25:19 serverB cyrus/master[31955][S7,I6,C1]: about to exec /usr/sbin/
     ctl_cyrusdb
Feb  1 19:25:19 serverB cyrus/master[4799][S7,I7,C1]: process 31955 exited, status 0
Feb  1 20:12:15 serverB ntpd[2933][S5,I7,C2]: kernel time sync status change 0001
Feb  2 13:30:51 serverB mysqld[2754][S2,I1,C6]: Feb  2 13:30:51 serverB mysqld[2754]:
     110202 13:30:51  InnoDB: Starting shutdown...
Feb  2 13:30:51 serverB mysqld[2754][S2,I4,C3]: 110202 13:30:51 [Note] /usr/sbin/mysqld:
     Normal shutdown
```

Listing 10.3 Server C: mocked-up event log messages

```
Feb  6 00:57:30 serverC kernel: [1069432.548709] md[S7,I5,C2]: data-check of RAID array
    md5
Feb  6 00:57:30 serverC kernel: [1069432.548715] md[S7,I7,C1]: minimum _guaranteed_
    speed: 1000 KB/sec/disk.
Feb  6 00:57:30 serverC kernel: [1069432.548724] md[S7,I6,C1]: using maximum available
    idle IO bandwidth (but not more than 200000 KB/sec) for data-check.
Feb  6 00:57:30 serverC kernel: [1069432.548735] md[S7,I7,C1]: using 128k window, over a
    total of 979840 blocks.
Feb  6 00:57:30 serverC kernel: [1069432.551279] md[S7,I7,C1]: delaying data-check of md6
     until md5 has finished (they share one or more physical units)
Feb  6 00:57:30 serverC kernel: [1069433.022527][S7,I7,C1]  --- wd:2 rd:2
Feb  6 00:57:30 serverC kernel: [1069433.022527][S7,I7,C1]  disk 0, wo:0, o:1, dev:sdb2
Feb  6 00:57:30 serverC kernel: [1069433.022527][S7,I7,C1]  disk 1, wo:0, o:1, dev:sda2
Feb  6 00:57:30 serverC kernel: [1069433.022527][S7,I7,C1] RAID1 conf printout:
Feb  6 00:58:00 serverC kernel: [1069462.353975] md[S7,I5,C2]: md5: data-check done
Feb  6 12:28:53 serverC apache2[8472][S4,I5,C4]: [06/Feb/2011 12:28:52 20583] [error] SSL
     handshake failed: HTTP spoken on HTTPS port; trying to send HTML error page (OpenSSL
     library error follows)
Feb  6 12:28:53 serverC apache2[8472][S4,I5,C4]: [06/Feb/2011 12:28:52 20583] [error]
    OpenSSL: error:1407609C:SSL routines:SSL23_GET_CLIENT_HELLO:http request [Hint:
    speaking HTTP to HTTPS port!?]
Feb  6 13:50:57 serverC kernel: [511040.208020] hub 1-1:1.0[S7,I7,C3]: activate --> -19
Feb  6 13:50:57 serverC kernel: [511040.208092] usb 1-1[S7,I7,C3]: USB disconnect,
    address 3
Feb  6 13:50:58 serverC kernel: [511040.320035] usb 1-1[S7,I7,C3]: new full speed USB
    device using uhci_hcd and address 5
Feb  6 13:50:58 serverC kernel: [511040.440024] usb 1-1[S3,I4,C6]: device descriptor read
    /64, error -71
Feb  6 13:50:58 serverC kernel: [511040.664035] usb 1-1[S3,I4,C6]: device descriptor read
    /64, error -71
Feb  6 13:50:59 serverC kernel: [511041.848038] usb 1-1[S3,I4,C6]: device not accepting
    address 5, error -71
Feb  6 13:51:00 serverC kernel: [511042.368098] hub 1-0:1.0[S3,I4,C6]: unable to
    enumerate USB device on port 1
Feb  7 11:19:20 serverC apache2[16971][S7,I7,C2]: example-client.net.nz - - [07/Feb
    /2011:11:19:20 +1300] "GET / HTTP/1.1" 200 2784 "-" "Mozilla/5.0 (Macintosh; U; Intel
     Mac OS X 10_6_6; en-us) Apple WebKit/533.19.4 (KHTML, like Gecko) Version/5.0.3
    Safari/533.19.4"
Feb  7 11:19:20 serverC apache2[16971][S7,I7,C2]: example-client.net.nz - - [07/Feb
    /2011:11:19:20 +1300] "GET /css/main.css HTTP/1.1" 200 2669 "http://example.example.
    com/" "Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_6_6; en-us) AppleWebKit/533.19.4
    (KHTML, like Gecko) Version/5.0.3 Safari/533.19.4"
Feb  7 11:19:20 serverC apache2[16971][S7,I7,C2]: example-client.net.nz - - [07/Feb
    /2011:11:19:22 +1300] "GET /images/bg_guests.gif HTTP/1.1" 200 91 "http://example.
    example.com/" "Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_6_6; en-us) AppleWebKit
    /533.19.4 (KHTML, like Gecko) Version/5.0.3 Safari/533.19.4"
Feb  7 11:19:22 serverC apache2[16971][S7,I7,C2]: example-client.net.nz - - [07/Feb
    /2011:11:19:22 +1300] "GET /images/heading_guests.gif HTTP/1.1" 200 2241 "http://
    example.example.com/" "Mozilla/5.0  (Macintosh; U; Intel Mac OS X 10_6_6; en-us)
    AppleWebKit/533.19.4 (KHTML, like Gecko) Version/5.0.3 Safari/533.19.4"
Feb  7 11:19:24 serverC apache2[16971][S7,I7,C2]: example-client.net.nz - - [07/Feb
    /2011:11:19:23 +1300] "GET /images/bg_guests_bottom.gif HTTP/1.1" 200 475 "http://
    example.example.com/" "Mozilla/5. 0 (Macintosh; U; Intel Mac OS X 10_6_6; en-us)
    AppleWebKit/533.19.4 (KHTML, like Gecko) Version/5.0.3 Safari/533.19.4"
Feb  7 11:19:24 serverC apache2[16971][S3,I6,C2]: example-client.net.nz - - [07/Feb
    /2011:11:19:23 +1300] "GET /images/bg_uplight.jpg HTTP/1.1" 404 216 "http://example.
    example.com/" "Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_6_6; en-us) AppleWebKit
    /533.19.4 (KHTML, like Gecko) Version/5.0.3 Safari/533.19.4"
Feb  7 14:59:27 serverC kernel: [   28.704024][S4,I6,C5] Clocksource tsc unstable (delta
    = -334799556 ns)
Feb  7 15:10:35 serverC kernel: [   696.379037][S7,I5,C3] XFS mounting filesystem dm-0
Feb  7 15:10:35 serverC kernel: [   696.880997][S7,I6,C2] Ending clean XFS mount for
    filesystem: dm-0
```



(a) Server A; impact weight "0"

(b) Server B; impact weight "0.6"

(c) Server C; impact weight "0.9"

Figure 10.1: Outcome-mockup values sorted by **I**mpact, then **S**everity, then **C**ertainty

Chapter 11

Appendix C: Victoria University of Wellington Human Ethics Policy

This is an excerpt from Section 4.7 of the VUW HEP [94], under which the survey was approved without a formal Ethics Committee hearing.

"**(a)** Research in which the subject's participation is restricted to the completion of a written questionnaire in a manner not requiring the disclosure of the subject's identity, and which meets the criteria for questionnaires in section 4.7(b), may be approved in writing by the Head of the School...

(b) The questionnaire must:

(i) Be totally anonymous (responses should be returned anonymously and there should be no coding or other means of identifying respondents from the response);

(ii) Not contain questions on sensitive topics (e.g. sexual practices, drug taking, illegal activities);

(iii) Be designed to meet the research goals set;

(iv) In the case of student projects, be subject to appropriate supervision;

(v) Normally state the purpose of the questionnaire, the use to which the results will be put, the disposal of the questionnaire forms, and the fact that the questionnaire is anonymous."

Chapter 12

Appendix D: Raw survey results

The horizontal axis is “participant/form”, the vertical axis is “response item”. Response item seven has two caveats: participants who appear to have reversed the rankings (i.e. used “10” to indicate the most important message) are noted with red cells, while the presence of optional comments is indicated with yellow cells. These optional comments were not included for brevity as well as consistency reasons: only two participants filled in *all* the comment spaces for response item seven. Any author comments are in green cells.

Form →	1	2	3	4	5	6	7	8	9	10	11
Response item ↓											
1a											
1b									1		
1c	1							1			
1d				1							
1e		1	1		1	1	1			1	1
1f											
1g											
2	6	5	4	5	4	5	7	4	4	4	2
3	4	2	5	5	3	7	5	4	5	2	4
4a	1	1	1	1		1	1	1	1		
4b	1	1	1	1	1			1	1	1	1
4c				1		1	1	1	1		
4d		1					1	1	1		
4e	1		1				1		1		
4f		1	1	1			1	1	1		1
4g					"Program debug output"					"Protocol daemons on routers, error alerts from interfaces"	
4h											
5a	1	1		1			1	1	1		1
5b		1	1	1	1	1		1	1	1	1
5c	1	1	1	1	1	1	1	1	1	1	1
5d	1	1	1	1			1	1	1	1	1
5e				1				1			
5f											
6a											
6b										1	
6c					1	1			1		
6d	1	1	1	1			1	1			1
Likely reversed ranking for item 7							1		1	1	

Table 12.1: Raw survey results, page 1

Form →	1	2	3	4	5	6	7	8	9	10	11
7a	10	1	2	8	9			9	5		
7b	4	6	3	4	4		9	6	9	3	
7c	7	8	9	9	8		8	2	6	10	
7d	6	5	5	5	5			7	1		
7e	1	10	8	1	1		10	1	3	8	
7f	9	2	1	10	10		1	10	6	1	
7g	8	9	10	3	3		7	8	5		
7h	2	7	6	2	2			4	4		
7i	5	4	7	7	6			5	5	5	
7j	3	3	4	6	7			3	2	7	
8	4	5	4	5	4		4	4	6	4	5
9	1	1	1	1	1	1	1	1	1	1	1

Table 12.2: Raw survey results, page 2

Form →	12	13	14	15	16	17	18	19	20	21	22
Response item ↓											
1a						1					
1b											
1c	1								1		
1d		1	1							1	
1e				1	1		1	1			1
1f											
1g											
2	4	6	6	5	5	1	4	4	7	7	3
3	5	5	4	5	4	5	4	5	6	7	5
4a	1	1	1			1	1				
4b	1		1		1	1	1	1			1
4c	1				1	1	1			1	
4d	1					1	1				
4e	1	1			1		1		1	1	
4f	1		1		1	1	1	1		1	1
4g				"Network events, e.g. link up/down"							
4h											
5a	1	1		1	1	1	1				
5b	1	1	1	1		1	1	1	1		
5c	1	1	1	1	1	1	1	1		1	1
5d	1	1	1	1	1	1	1	1			1
5e	1	1					1				
5f											
6a											
6b											
6c										1	
6d	1	1	1	1	1	1	1	1	1		1
Likely reversed ranking for item 7			1	1							1

Table 12.3: Raw survey results, page 3

Form →	12	13	14	15	16	17	18	19	20	21	22
7a	9	9	3	1	9				10	9	2
7b	6	8	4	9	4				4	4	3
7c	1	2	5	8	5				1	1	10
7d	8	7	2	6	6				3	7	9
7e	2	1	9	10	7				2	2	8
7f	10	10	1	3	9				7	10	1
7g	4	6	6	7	3				5	5	4
7h	7	5	8	2	2				9	3	6
7i	3	4	7	5	1				8	8	7
7j	5	3	10	4	10				6	6	5
8	7	6	6	4	4	4	4		6	4	5
9	1	1	1	1	1	1	1	1	1	1	1

Table 12.4: Raw survey results, page 4

Form →	23	24	25	26	27	28	29	30	31	32	33
Response item ↓						Duplicate of 27					
1a											
1b											
1c					1		1	1			
1d		1	1								1
1e	1			1					1	1	
1f											
1g											
2	7	2	7	2	6		3	4	4	3	3
3	6	4	3	3	6		5	4	5	2	4
4a	1	1	1	1			1	1	1	1	1
4b	1	1	1	1			1	1	1		1
4c	1	1	1	1			1	1	1		1
4d				1			1	1	1		1
4e	1		1	1					1		1
4f		1	1	1			1	1		1	1
4g											
4h											
5a	1	1		1	1		1			1	1
5b	1	1	1	1	1		1	1			
5c	1	1	1	1	1		1	1			1
5d	1		1	1	1		1	1		1	
5e				1			1	1			
5f									“Experience”		
6a											
6b											
6c				1							
6d	1	1	1		1		1	1	1	1	1
Likely reversed ranking for item 7	1		1							1	

Table 12.5: Raw survey results, page 5

Form →	23	24	25	26	27	28	29	30	31	32	33
7a	2	3	3	10		Duplicate of 27	8	9	9	3	10
7b	10	5	5	5			4	3	1	4	9
7c	10	4	2	1			3	2	3	10	3
7d	7	2	4	7			6	8	5	2	8
7e	9	1	9	2			1	4	7	8	1
7f	10	9	1	6			7	10	10	1	7
7g	7	6	8	8			5	1	4	9	5
7h		10	10	9			2	6	8	5	2
7i	4	7	7	4			10	5	6	6	4
7j		8	6	3			9	7	2	7	6
8	5	5	4	7	4		5	6	7	5	5
9	1	1	1	1	1		1	1	1	1	1

Table 12.6: Raw survey results, page 6

Form →	34	35	36	37	38	39	40	41	42	43	44
Response item ↓											
1a											
1b											
1c									1		
1d			1	1	1		1				
1e	1	1				1		1		1	1
1f											
1g											
2	3	4	6	5	6	4	3	5	5	5	2
3	3	4	5	4	4	3	4	5	5	5	4
4a	1		1	1		1			1	1	
4b	1	1	1	1	1	1			1	1	1
4c		1		1	1	1	1		1		1
4d				1		1					
4e		1		1	1	1	1				
4f				1	1	1	1		1	1	1
4g					“Firewall/IDS”			“Specific extractions based on the system task”			
4h								1			
5a		1		1	1		1		1		
5b	1		1	1	1	1	1		1		
5c	1	1			1	1	1		1	1	
5d			1	1	1				1	1	1
5e		1		1	1	1			1	1	
5f								“Revavalence to issue Z experance. ie. problem occur on x log message received at x”			
6a											
6b											1
6c							1		1		
6d	1	1	1	1		1		1		1	
Likely reversed ranking for item 7											

Table 12.7: Raw survey results, page 7

Form →	34	35	36	37	38	39	40	41	42	43	44
7a	9	8		10	10	10			9	2	3
7b	7	3		6	3	7			3	6	5
7c	1	9		4	9	6			2	3	9
7d	10	4		9	4	9			8	10	8
7e	2	1		1	1	1			1	5	4
7f	4	10		8	5	8			10	1	6
7g	5	5		7	8	2			4	8	7
7h	3	7		5	2	3			7	7	2
7i	6	2		6	6	4			5	9	10
7j	8	6		2	7	5			6	4	1
											random ranking?
8	5	4	6	7	6	4	5	2	6	7	2
9	1	1	1	1	1	1	1	1	1	1	1

Table 12.8: Raw survey results, page 8

Form →	45	46	47	48	49	50
Response item ↓						
1a						1
1b					1	
1c				1		
1d			1			
1e	1	1				
1f						
1g						
2	7	3	3	7	4	2
3	6	3	3	4	5	3
4a		1	1	1	1	1
4b			1	1	1	1
4c		1	1	1	1	1
4d			1		1	1
4e		1	1		1	
4f	1		1	1	1	1
4g	“ports, interfaces, services, links”		“network utilization alerts, latency/loss/jitter alerts”			
4h						
5a	1	1	1		1	
5b	1	1	1	1	1	1
5c	1		1	1	1	1
5d	1	1	1	1	1	1
5e			1		1	
5f						
6a						
6b				1		
6c						1
6d	1	1	1		1	
Likely reversed ranking for item 7						

Table 12.9: Raw survey results, page 9

Form →	45	46	47	48	49	50
7a					10	
7b					6	
7c					3	
7d					8	
7e					1	
7f					9	
7g					2	
7h					4	
7i					5	
7j					7	
8	6	3	4		4	6
9	1	1	1	1	1	1

Table 12.10: Raw survey results, page 10

Chapter 13

GPL License

This license is included here as a legal requirement, given that GPL-licensed code has been included in this document.

GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Lesser General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow. TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on

each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change. b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License. c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or, b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or, c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any

further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. END OF TERMS AND CONDITIONS How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

one line to give the program's name and an idea of what it does. Copyright (C) yyyy name of author

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) year name of author Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'. This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than 'show w' and 'show c'; they could even be mouse-clicks or menu items–whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program 'Gnomovision' (which makes passes at compilers) written by James Hacker.

signature of Ty Coon, 1 April 1989 Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License.

Bibliography

[1] R. Mitchell, B. Agle, and D. Wood, “Toward a theory of stakeholder identification and salience: defining the principle of who and what really counts,” *Academy of Management Review*, vol. 22, no. 4, pp. 853–86, 1997.

[2] T. M. Barron, R. H. L. Chiang, and V. C. Storey, “A semiotics framework for information systems classification and development,” *Decision Support Systems*, vol. 25, no. 1, pp. 1–17, February 1999.

[3] R. K. Stamper, “Signs, Information, Norms and Systems,” in *Signs at Work*, B. Holmqvist, P. B. Andersen, H. Klein, and R. Posner, Eds. Berlin: De Gruyter, 1996, pp. 349–397.

[4] P. Barr, “User-Interface Metaphors in Theory and Practice,” Master’s thesis, Victoria University of Wellington, December 2003.

[5] G. Doeben-Henisch and M. F. Wagner, “Validation within Safety Critical Systems Engineering from a Computational Semiotics Point of View,” in *Proc. IEEE Africon2007 International Conference*. Piscataway, NJ, USA: IEEE Computer Society, 2007.

[6] P. Cobley and L. Jansz, *Introducing Semiotics*. Cambridge, England: Icon Books Ltd., 1999.

[7] C. E. Shannon, “Communication in the Presence of Noise,” *Proceedings of the IRE*, vol. 37, no. 1, pp. 10–21, 1949.

[8] W. Weaver, “Recent Contributions to The Mathematical Theory of Communication,” *The Mathematical Theory of Communication*, vol. 1, pp. 93–117, 1949.

[9] C. Matheus, M. Kokar, and K. Baclawski, “A Core Ontology for Situation Awareness,” in *Proc. FUSION’03*, Cairns, Queensland, Australia, 2003, pp. 545–552.

[10] S. Wallin, V. Leijon, and L. Landén, "Statistical analysis and prioritisation of alarms in mobile networks," *International Journal of Business Intelligence and Data Mining*, vol. 4, no. 1, pp. 4–21, 2009.

[11] *RFC 5424: The Syslog Protocol*, Network Working Group Std., 2009. Available at: http://tools.ietf.org/html/rfc5424

[12] K. Prasad, "The Glasgow Coma Scale: A Critical Appraisal of its Clinimetric Properties," *Journal of Clinical Epidemiology*, vol. 49, no. 7, pp. 755–763, July 1996.

[13] OpenVPN project. (Retrieved 2010-07-16). Available at: http://openvpn.net

[14] S. Few, *Information Dashboard Design: The Effective Visual Communication of Data.* Sebastopol, CA, USA: O'Reilly Media Inc, 2006, iSBN 978-0-596-10016-2.

[15] Z. Li, J. Taylor, E. Partridge, Y. Zhou, W. Yurcik, C. Abad, J. J. Barlow, and J. Rosendale, "UCLog: A unified, correlated logging architecture for intrusion detection," in *Proc. 12th ICTSMA*, 2004.

[16] N. Xianjun, "Design and implementation of WEB log mining system," in *Proc. 2009 International Conference on Computer Engineering and Technology.* Piscataway, NJ, USA: IEEE Computer Society, January 2009, pp. 425–427.

[17] W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordan, "Online System Problem Detection by Mining Patterns of Console Logs," in *Proc. 9th IEEE International Conference on Data Mining.* IEEE Computer Society, December 2009, pp. 588–597.

[18] K. Yamanishi and Y. Maruyama, "Dynamic Syslog Mining for Network Failure Monitoring," in *Proc. 11th ACM SIGKDD international conference on knowledge discovery in data mining.* ACM New York, NY, USA, 2005, pp. 499 – 508.

[19] LogWatch. (Retrieved 2010-06-22). Available at: http://www.logwatch.org

[20] Logcheck. (Retrieved 2010-06-22). Available at: http://logcheck.org

[21] Debian Popularity Contest. (Retrieved 2010-06-23). Available at: http://popcon.debian.org

[22] NMS. (Retrieved 2010-06-23). Available at: http://www.nimsoft.com

[23] Snare Server. (Retrieved 2010-06-24). Available at: http://www.intersectalliance.com

[24] Splunk. (Retrieved 2010-06-24). Available at: http://www.splunk.com

[25] M. F. Buckley and D. P. Siewiorek, "VAX/VMS Event Monitoring and Analysis," in *FTCS, Computing Digest of Papers*, June 1995, pp. 414–423.

[26] M. F. Buckley, "Computer Event Monitoring and Analysis," Ph.D. dissertation, Carnegie-Mellon University, Pittsburgh, Pennsylvania, USA, May 1992.

[27] K. Kent and M. Souppaya, *Guide to Computer Security Log Management (SP800-92)*, Information Technology Laboratory Std., September 2006.

[28] S. Tricaud and P. Saadé, "Applied parallel coordinates for logs and network traffic attack analysis," *Journal in Computer Virology*, vol. 6, no. 1, pp. 1–29, January 2010.

[29] N. Hammoud, "Decentralized Log Event Correlation Architecture," in *Proc. International Conference on Management of Emergent Digital EcoSystems.* ACM New York, NY, USA, 2009, pp. 480–482.

[30] D. Caragea and V. Honavar, "Learning Classifiers from Distributed Data Sources," *Encyclopedia of Database Technologies and Applications*, vol. 2nd edition, p. unknown, 2008.

[31] J. Saltzer and M. Schroeder, "The Protection of Information in Computer Systems," *Proceedings of the IEEE*, vol. 63, no. 9, pp. 1278–1308, 1975.

[32] S. Etalle, F. Massacci, and A. Yautsiukhin, "The Meaning of Logs," in *Trust, Privacy and Security in Digital Business: TrustBus 2007*, ser. LNCS, C. Lambrinoudakis, G. Pernul, and A. M. Tjoa, Eds., vol. 4657. Springer, August 2007, pp. 145–154.

[33] D. Tu, R. Chen, Z. Du, and Y. Liu, "A Method of Log File Analysis for Test Oracle," in *Proc. 2009 International Conference on Scalable Computing and Communications; Eighth International Conference on Embedded Computing.* Washington, DC, USA: IEEE Computer Society, 2009, pp. 351–354.

[34] E. Yoneki and J. Bacon, "Ubiquitous Computing: Challenges in Flexible Data Aggregation," in *Embedded and Ubiquitous Computing*, ser. LNCS, L. T. Yang, Ed., vol. 3824. Springer, November 2005, pp. 1189–1200.

[35] M. A. GonÇalves, M. Luo, R. Shen, M. F. Ali, and E. A. Fox, "An XML Log Standard and Tool for Digital Library Logging Analysis," in *ECDL 2002*, ser. LNCS, M. Agosti and C. Thanos, Eds., vol. 2458. Springer, 2002, pp. 129–143.

[36] S. M. Serra da Cruz, M. L. M. Campos, P. F. Pires, and L. M. Campos, “Monitoring E-Business Web Services Usage through a Log Based Architecture,” in *Proc. IEEE Conference ICWS*. IEEE Computer Society, 2004, pp. 61–69.

[37] V. Stathopoulosa, P. Kotzanikolaoua, and E. Magkos, “Secure log management for privacy assurance in electronic communications,” *Computers & Security*, vol. 27, no. 7-8, pp. 298–308, December 2008.

[38] B. R. Waters, D. Balfanz, G. Durfee, and D. K. Smetters, “Building an Encrypted and Searchable Audit Log,” in *ISOC Network and Distributed System Security Symposium*. The Internet Society, February 2004.

[39] S. Weber, “Harnessing Pseudonyms with Implicit Attributes for Privacy-Respecting Mission Log Analysis,” in *Intelligent Networking and Collaborative Systems*. IEEE Computer Society, November 2009, pp. 119–126.

[40] C. Flack and M. J. Atallah, “Better Logging through Formality,” in *Proc. Recent Advances in Intrusion Detection*, ser. LNCS, H. Debar, L. Mé, and F. Wu, Eds., vol. 1907. Springer, 2000, pp. 1–16.

[41] M. Mansouri-Samani and M. Sloman, “GEM: a generalized event monitoring language for distributed systems,” *Distributed Systems Engineering*, vol. 4, no. 2, pp. 96–108, June 1997.

[42] H. Saneifar, S. Bonniol, A. Laurent, P. Poncelet, and M. Roche, “Terminology Extraction from Log Files,” in *Proc. Database and Expert Systems Applications - 20th International Conference*, ser. LNCS, S. S. Bhowmick, J. Küng, and R. Wagner, Eds., vol. 5690. Springer, 2009, pp. 769–776.

[43] *X.733: Information Technology - Open Systems Interconnection - Systems Management: Alarm Reporting Function*, CCITT Std., 1992.

[44] Rsyslog. (Retrieved 2010-11-17). Available at: http://www.rsyslog.com

[45] D. Schreckenghost, C. Martin, and C. Thronesbery, “Specifying Organizational Policies and Individual Preferences for Human-Software Interaction,” in *Etiquette for Human-Computer Work, Papers from the AAAI Fall Symposium. Technical Report FS-02-02*. AAAI Press, 2002.

[46] A. M. Colman, “Salience and focusing in pure coordination games,” *Journal of Economic Methodology*, vol. 4, no. 1, pp. 61–81, 1997.

[47] C Language Integrated Production System (CLIPS) Programming Guide. (Retrieved 2010-12-07). Available at: http://www.csie.ntu.edu.tw/~sylee/courses/clips/bpg/node5.4.10.1.html

[48] J. R. Eiser, "Categorization, cognitive consistency and the concept of dimensional salience," *European Journal of Social Psychology*, vol. 1, pp. 435–454, 1971.

[49] G. Teasdale and B. Jennett, "Assessment of coma and impaired consciousness: a practical scale," *Lancet*, vol. 304, no. 7872, pp. 81–84, July 1974.

[50] B. Edvardsson and M. Linden, "A method for evaluation of metric properties of response scales," *Quality and Quantity*, vol. 10, pp. 241–249, 1976.

[51] T. Cook and D. T. Campbell, *Quasi-Experimentation: Design and Analysis Issues for Field Settings.* Chicago: Rand McNally, 1979.

[52] L. Bunch, M. Breedy, J. M. Bradshaw, M. Carvalho, N. Suri, A. Uszok, J. Hansen, M. Pechoucek, and V. Marik, "Software Agents for Process Monitoring and Notification," in *Proc. 2004 ACM symposium on Applied computing.* ACM New York, NY, USA, March 2004, pp. 94–100.

[53] Information Technology Infrastructure Library (ITIL). (Retrieved 2010-08-12). Available at: http://www.itil-officialsite.com/

[54] S. Littlejohn, *Theories of human communication*, 7th ed. Belmont, California, USA: Wadsworth/Thomson Learning, 2002.

[55] C. S. De Souza and C. F. Leitao, *Semiotic Engineering Methods for Scientific Research in HCI*, ser. Synthesis Lectures on Human-centred Informatics, J. M. Carroll, Ed. Morgan & Claypool Publishers, 2009.

[56] T. Atwood, "The role of pattern databases in sequence analysis," *Briefings in Bioinformatics*, vol. 1, no. 1, pp. 45–59, 2000. Available at: http://bib.oxfordjournals.org/cgi/content/abstract/1/1/45

[57] W. E. Willison, "Data Logging in Power Generating Stations," *British Radio and Electronic Engineer*, vol. 25, no. 6, pp. 559–570, June 1963.

[58] R. Molloy and R. Parasuraman, "Monitoring an Automated System for a Single Failure: Vigilance and Task Complexity Effects," *Human Factors*, vol. 38, no. 2, pp. 311–322, 1996.

[59] E. Pinheiro, W. Weber, and L. Barroso, "Failure Trends in a Large Disk Drive Population," in *Proc. 5th USENIX Conference on File and Storage Technologies (FAST'07)*, San Jose, CA, USA, 2007. Available at: http://labs.google.com/papers/disk_failures.html

[60] J. Reason, *Human Error.* New York, USA: Cambridge University Press, 1990.

[61] R. K. Stamper, K. Liu, M. Hafkamp, and Y. Ades, "Understanding the roles of signs and norms in organizations - a semiotic approach to information systems design," *Behaviour & Information Technology*, vol. 19, no. 1, pp. 15–27, 2000.

[62] P. Radford, A. Linton, and I. Welch, "Event log messages as a human interface," in *Proc. OZCHI 2010*, Accepted for publication.

[63] H. Tsunoda, T. Maruyama, K. Ohta, Y. Waizumi, G. M. Keeni, and Y. Nemoto, "A Prioritized Retransmission Mechanism for Reliable and Efficient Delivery of Syslog Messages," in *Proc. Communication Networks and Services Research Conference*, 2009, pp. 158 – 165.

[64] M. T. Rose, *The Simple Book: An introduction to management of TCP/IP-based internets*, ser. Prentice Hall Series in Innovative Technology, D. R. Allison, D. J. Farber, and B. D. Shriver, Eds. New Jersey, USA: Prentice-Hall, Inc., 1991.

[65] M. Nagappan, K. Wu, and M. A. Vouk, "Efficiently Extracting Operational Profiles from Execution Logs Using Suffix Arrays," in *Proc. Software Reliability Engineering.* IEEE Computer Society, November 2009, pp. 41–50.

[66] F. Facca and P. Lanzi, "Mining interesting knowledge from weblogs: a survey," *Data & Knowledge Engineering*, vol. 53, no. 3, pp. 225–241, 2005, elsevier.

[67] R. Du, R. Safavi-Naini, and W. Susilon, "Web filtering using text classification," in *Proc. 11th IEEE International Conference on Networks*, 2003, pp. 325–330. Available at: http://ro.uow.edu.au/infopapers/166

[68] Debian Linux Project. (Retrieved 2010-06-21). Available at: http://www.debian.org

[69] Berkeley Internet Name Daemon. (Retrieved 2010-07-29). Available at: http://www.isc.org/software/bind

[70] T. Moser, H. Roth, S. Rozsnyai, R. Mordinyi, and S. Biffl, "Semantic Event Correlation Using Ontologies," in *On the Move to Meaningful Internet Systems,*

ser. LNCS, R. Meersman, T. Dillon, and P. Herrero, Eds., vol. 5871. Springer, November 2009, pp. 1087–1094.

[71] A. Boury-Brisset, "Ontology-based Approach for Information Fusion," in *Proc. 6th International Conference on Information Fusion*, 2003, pp. 522–529.

[72] R. Sterritt, "Towards Autonomic Computing: Effective Event Management," in *Proc. 27th Annual NASA Goddard/IEEE Software Engineering Workshop.* Maryland, USA: IEEE Computer Society, December 2002, pp. 40–47.

[73] A. Zimmermann, R. Sahay, R. Fox, and A. Polleres, "Heterogeneity and Context in Semantic-Web-Enabled HCLS Systems," in *On the Move to Meaningful Internet Systems*, ser. LNCS, T. Meersman, R.and Dillon and P. Herrero, Eds., vol. 5871. Springer, November 2009, pp. 1165–1182.

[74] USENIX LANL supercomputer event logs (messages.sdb file). (Retrieved 2010-09-24). Available at: http://cfdr.usenix.org/data.html,0807080346.tar.gzfile

[75] Munin. (Retrieved 2010-04-22). Available at: http://munin-monitoring.org

[76] S. Franklin and A. Graesser, "Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents," in *Intelligent Agents III Agent Theories, Architectures, and Languages*, ser. LNCS, vol. 1193. Springer, 1997, pp. 21–35.

[77] M. Luck and M. d'Inverno, "A Formal Framework for Agency and Autonomy," in *Proceedings of the First International Conference on Multi-Agent Systems*, 1995, pp. 254–260.

[78] J. M. Bradshaw, P. Beautement, M. R. Breedy, L. Bunch, S. V. Drakunov, P. J. Feltovich, R. R. Hoffman, R. Jeffers, M. Johnson, S. Kulkarnt, L. Lott, A. K. Raj, N. Suri, and A. Uszok, *Intelligent Technologies for Information Analysis: Advances in Agents, Data Mining, and Statistical Learning.* Springer, 2004, ch. 12: Making Agents Acceptable to People, pp. 355–400.

[79] J. R. Saul, *The Doubter's Companion: A Dictionary of Aggressive Common Sense.* Toronto, Canada: Penguin Books, 1995.

[80] LinuxForDevices. (Retrieved 2011-02-07). Available at: http://www.linuxfordevices.com/c/a/News/Linux-kernel-developers-have-tripled-in-number/

[81] H. Subramanian, S. Ramamoorthy, P. Stone, and B. J. Kuipers, "Designing Safe, Profitable Automated Stock Trading Agents Using Evolutionary Algorithms," in

GECCO '06: Proc. 8th annual conference on Genetic and evolutionary computation. ACM New York, NY, USA, 2006, pp. 1777–1784.

[82] A. Sherstov and P. Stone, "Three Automated Stock-Trading Agents: A Comparative Study," in *Agent-Mediated Electronic Commerce VI*, ser. LNCS, P. Faratin and J. Rodriguez-Aguilar, Eds. Springer, 2005, vol. 3435, pp. 173–187.

[83] Y. Zhai, A. Hsu, and S. Halgamuge, "Combining News and Technical Indicators in Daily Stock Price Trends Prediction," in *Advances in Neural Networks, ISNN 2007*, ser. LNCS, D. Liu, S. Fei, Z. Hou, H. Zhang, and C. Sun, Eds., vol. 4493. Springer, 2007, pp. 1087–1096.

[84] Algorithms take control of Wall Street. (Retrieved 2011-01-11). Available at: http://arstechnica.com/tech-policy/news/2011/01/algorithms-take-control-of-wall-street.ars

[85] The stock market as a single, very big piece of multithreaded software. (Retrieved 2010-10-30). Available at: http://arstechnica.com/business/news/2010/10/you-say-stock-market-i-say-ginormous-multithreaded-application.ars

[86] LibreOffice. (Retrieved 2011-02-02). Available at: http://www.libreoffice.org/

[87] OpenOffice.org is Dead, Long Live LibreOffice – or, The Freedom to Fork, by Terry Hancock. (Retrieved 2010-10-06). Available at: http://www.freesoftwaremagazine.com/columns/openoffice_org_dead_long_live_libreoffice

[88] The Document Foundation. (Retrieved 2011-02-10). Available at: http://www.documentfoundation.org/foundation/

[89] P. Alreck and R. Settle, *The Survey Research Handbook*, 3rd ed. New York, USA: McGraw-Hill/Irwin, 2004.

[90] *RFC 3164: The BSD Syslog Protocol*, Network Working Group Std., 2001. Available at: http://tools.ietf.org/html/rfc3164

[91] S. A. Macskassy and F. Provost, "Intelligent Information Triage," in *Proc. 24th Annual International ACM SIGIR conference on Research and Development in Information Retrieval.* Louisiana, USA: ACM New York, NY, USA, September 2001, pp. 318–326.

[92] New Zealand Network Operators Group. (Retrieved 2010-05-12). Available at: http://www.nznog.org

[93] Y. Baruch and B. Holtom, "Survey response rate levels and trends in organizational research," *Human Relations*, vol. 61, no. 8, pp. 1139–1160, 2008.

[94] VUW Human Ethics Policy. (Retrieved 2010-10-27). Available at: http://policy.vuw.ac.nz/Amphora!~~policy.vuw.ac.nz~POLICY~000000000744.pdf

[95] Apache server documentation. (Retrieved 2011-03-03). Available at: http://httpd.apache.org/docs/1.3/howto/auth.html

[96] R. Ando, "Automated Log Analysis of Infected Windows OS Using Mechanized Reasoning," in *Neural Information Processing - 16th International Conference, ICONIP 2009*, ser. LNCS, C. S. Leung, M. Lee, and J. H. Chan, Eds., vol. 5864. Springer, December 2009, pp. 540–547.

[97] T. T. Y. Lin and D. P. Siewiorek, "Error Log Analysis: Statistical Modeling and Heuristic Trend Analysis," *IEEE Transactions on Reliability*, vol. 39, no. 4, pp. 419–432, 1990.

[98] A. Blackwell, *Your Wish Is My Command: Programming by Example*, H. Lieberman, Ed. London: Academic Press, 2001.

[99] E. Daniel, R. Lal, and G. Choi, "Warnings and Errors: A Measurement Study of a UNIX Server," in *Proc. 29th IEEE Int. Symposium on Fault-Tolerant Computing*, vol. 29, 1999.

[100] C. Gunther and W. van der Aalst, "A generic import framework for process event logs," in *Business Process Management Workshops.* Springer, 2006, pp. 81–92.

[101] D. Jonker, W. Wright, D. Schroh, P. Proulx, and B. Cort, "Information triage with TRIST," in *2005 International Conference on Intelligence Analysis.* Washington, DC, USA: Oculus Info, Inc., May 2005.

[102] J. Lewis, "IBM computer usability satisfaction questionnaires: psychometric evaluation and instructions for use," *International Journal of Human-Computer Interaction*, vol. 7, no. 1, pp. 57–78, 1995.

[103] LogSaw. (Retrieved 2010-07-23). Available at: http://logsaw.sourceforge.net/

[104] The Limitations of Server Log Files for Usability Analysis. (Retrieved 2010-07-23). Available at: http://www.boxesandarrows.com/view/the-limitations-of