# Challenging the Mean Time to Failure: Measuring Dependability as a Mean Failure Cost*

Ali Mili,
College of Computing Science,
New Jersey Institute of Technology,
Newark NJ 07102-1982
mili@cis.njit.edu

Frederick Sheldon
Oak Ridge National Laboratory
1 Bethel Road
Oak Ridge TN 37831
sheldonft@ornl.gov

## Abstract

*As a measure of system reliability, the mean time to failure falls short on many fronts: it ignores the variance in stakes among stakeholders; it fails to recognize the structure of complex specifications as the aggregate of overlapping requirements; it fails to recognize that different components of the specification carry different stakes, even for the same stakeholder; it fails to recognize that V& V actions have different impacts with respect to the different components of the specification. Similar metrics of security, such as MTTD (Mean Time to Detection) and MTTE (Mean Time to Exploitation) suffer from the same shortcomings. In this paper we advocate a measure of dependability that acknowledges the aggregate structure of complex system specifications, and takes into account variations by stakeholder, by specification components, and by V& V impact.*

## 1 Challenging the Mean Time to Failure

The Mean Time to Failure (MTTF) is a commonly accepted measure for system reliability. Similar metrics of security, such as the Mean Time To Detection (of a security vulnerability), abreviated by MTTD, and the Mean Time To Exploitation (of a security vulnerability), abbreviated by MTTE, have also been proposed [12]. In this paper we submit a tentative challenge to these metrics, by highlighting their shortcomings and suggesting appropriate remedies. Because MTTF is by far better known than MTTD and MTTE, we focus our discussion on measuring reliability, and argue that the same approach can be applied to other dimensions of dependability, such as security.

When we say that a system $S$ has an MTTF $M$, we mean that the mean time to the failure of the system with respect to some implicit specification $R$ is $M$. In doing so, we are usually making three implicit assumptions:

- *Independence of failure cost with respect to subspecifications*. A complex specification is typically the aggregate of many individual requirements/ subspecification[1]; the stakes attached to meeting each requirement vary from one requirement to another. Yet the MTTF makes no distinction between requirements; failing any requirement, whether it be a high stake or a low stake requirement, counts as a failure.

- *Independence with respect to stakeholders*. Typically the operation of a system involves many stakeholders, who have different stakes in the system meeting any given requirement. Yet the MTTF is not dependent on the stakeholder but exclusively on the system under observation.

- *Independence of failure probability with respect to subspecifications*. A complex specification is typically the aggregate of many individual requirements/ subspecifications. Verification and validation activities may give us higher levels of confidence in meeting some requirements than in meeting others. Yet the MTTF does not account for this variance, assuming that any failure with respect to any subspecification is a failure with respect to the whole specification.

We propose in this paper a measure of reliability (and safety) that takes into account the variance in failure cost

*This manuscript has been authored by a contractor of the U.S. Government under contract DE-AC05-00OR22725. Accordingly, the U.S. Government retains a nonexclusive, royalty-free licence to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes.

[1] We use the terms *requirement* and *subspecification* interchangeably, to refer to a component of a compound specification (the first term refers to a contents, whereas the second term refers to a form).

from one supspecification to another, the variance in failure probability from one subspecification to another, and the variance in failure impact from one stakeholder to another. We refer to this measure as the *Mean Failure Cost*, and abbreviate it by *MFC*.

## 2 A Motivating Example

To illustrate our ideas and motivate the interest of the proposed measure, we consider a simple example: We consider the specification of a sorting program, which is made up of two subspecifications: $Ord$, which specifies that the final array is sorted in say, increasing order; $Prm$, which specifies that the final array is a permutation of the initial array. We refer to the aggregate specification as $Sort$. Furthermore, we consider five stakeholders, who attach different stakes to whether a candidate sorting program meets these requirements:

- *Binary Searcher*. This stakeholder uses the output of the sorting program to do binary searches on the array. Binary Search works only if the array is sorted, and its behavior is unpredictable if the array is out of order by even one cell (if that cell is used as a pivot, the search may be dispatched to the wrong side of the array). We assume further that this stakeholder does not attach a great premium on whether the array has lost any of its original values, provided it is perfectly sorted.

- *The Median Finder*. This stakeholder uses the sorting program only to find the median of the original sequence. Once the array is sorted, this stakeholder retrieves the element in the middle of the array. This stakeholder does not place a very high premium on the array being perfectly ordered, for if some cells are out of order but are on the right side of the median, this does not affect the median. Also, this stakeholder does not place a very high premium on the array being a permutation of the initial array, for he is affected only if the cell he retrieves in the middle of the array turns out not to be an element of the initial array.

- *The Average Finder*. This stakeholder is interested in computing the average of the array. The only stake he has in the array being sorted is that when the array cells are added in increasing order, round off errors are minimized. He does have some stake in preserving the array cells, but is affected only if the sum of the array is altered, and the impact of the sum is diminished when we divide it by the size of the array.

- *The Linear Searcher*. This stakeholder places a high premium on the array being a permutation of the initial array, and places a relatively high premium on the array being sorted, because if not he may miss some elements of the array in his search.

- *The Brute Force Searcher*. This stakeholder places a very high premium on the array being a permutation of the initial array, but does not care at all whether the array is sorted, because his search algorithm scans the whole array anyway.

We quantify the varying stakes that stakeholders have in meeting requirements $Ord$ and $Prm$ by their corresponding failure costs, measured in dollars per hour of operation; see Figure 1. Now, we further assume that a sorting program that was written to satisfy this specification has been validated as follows:

- An inspection team has inspected the code line by line and found that the array is never altered expect by a code sequence that merely swaps two cells. The inspection team has concluded that we can be assured with a very high probability (0.9999) that the candidate sorting program satisfies specification $Prm$ over a period of operation $T$.

- A testing team has run the program using several test generation techniques, employing several diverse coverage criteria techniques, and the program has passed the tests successfully, with respect to an oracle that only tests whether the output is ordered (it is far too costly to check for $Prm$ at run time). In light of test results, the testing team estimates that the candidate sorting program satisfies specification $Ord$ with a probability of 0.99 over the same period of time $T$.

- Assuming independence of these two requirements (i.e. statistical independence of the events that the candidate program is correct with respect to these requirements), we estimate the probability of satisfying the aggregate specification $Sort$:

$$0.99 \times 0.9999 = 0.98991.$$

We record the result of these discussions in Figure 1.

Using this information, we propose to compute the mean failure cost, for each stakeholder, as the sum of failure costs with respect to the subspecifications, weighted by the probabilities of failing to satisfy these subspecifications. On the other hand, in order to compute the MTTF of this system with respect to $Sort$, we take two (outrageously) simplistic assumptions, namely that the failure density function is an exponential (more realistic for hardware than software), and that the failure rate is constant (usually it is bathtub shaped). Knowing that the system

| Stakeholders | $Ord$ | $Prm$ | $Sort$ |
|---|---|---|---|
| Binary Searcher | 10 | 2 | |
| Median Finder | 4 | 4 | |
| Average Finder | 1 | 3 | |
| Linear Searcher | 7 | 10 | |
| Brute Force Searcher | 0 | 10 | |
| Probability of Success | 0.99 | 0.9999 | 0.98991 |
| Probability of Failure | 0.01 | 0.0001 | 0.01009 |

**Figure 1. Variance in Stakes and Probabilities**

| Stakeholders | $Ord$ | $Prm$ | $Sort$ | MFC $/Hr |
|---|---|---|---|---|
| Binary Searcher | 10 | 2 | | 0.1002 |
| Median Finder | 4 | 4 | | 0.0404 |
| Average Finder | 1 | 3 | | 0.0103 |
| Linear Searcher | 7 | 10 | | 0.0710 |
| B/F Searcher | 0 | 10 | | 0.0010 |
| P(Success) | 0.99 | 0.9999 | 0.98991 | |
| P(Failure) | 0.01 | 0.0001 | 0.01009 | |
| MTTF (Hrs) | | | 98.6072 | |

**Figure 2. MFC vs. MTTF**

has probability $0.98991$ of operating failure-free for a duration $T$ allows us to derive the failure rate, from which we infer the MTTF under the cited assumptions [11]. The result is given in Figure 2.

We find that the MTTF is 98.6072 hours, while the mean failure cost is 0.1002 $/ hour for the Binary Searcher, 0.0404 $/ hour for the Median Finder, 0.0103 $/ hour for the Average Finder, 0.0710 $/ hour for the Linear Searcher, and 0.0010 $/ hour for the Brute Force Searcher.

Even though the $MTTF$ is the same for all stakeholders, the mean failure cost varies widely, by two orders of magnitude, from one stakeholder to another, due to the different stakes they have in meeting the various requirements, and the different probabilities we have for meeting each requirement. We argue that the mean failure cost is a more meaningful measure to each stakeholder than the MTTF.

# 3 Background for a Systematic Approach

In the previous section, we have used a simple example to illustrate what we mean by mean failure cost, and to contrast it to the traditional MTTF metric. To compute the mean failure cost for a stakeholder, we have simply taken the failure costs attached by the stakeholder to each component of the specification, and have prorated them with the probabilities of failing each component of the specification. In doing so, we have made some assumptions, which are not in fact borne out:

- *That all the stakeholders share the same decomposition of the overall specification*. Using again the example of the sorting program, we can imagine a stakeholder who places a high premium on meeting specification $Sort$ for large arrays, but not for small arrays; formulating this would require a different decomposition of $Sort$ from that which we discussed in the previous section.
- *That probabilities are multiplicative*. The tentative formula of MFC assumes that the events of refining two subspecifications, say $S_1$ and $S_2$, are statistically independent; i.e. the probability of refining them simultaneously is the product of the probabilities associated with refining each.
- *That costs are additive*. The tentative formula of MFC asumes that the costs of failing to satisfy two subspecifications, say $S_1$ and $S_2$, are additive; i.e. the cost of failing them simultaneously is the sum of the costs of failing each one.

As a first step towards a theory for defining and estimating mean failure costs, we must understand the structure of specifications. In this section, we discuss the refinement structure and the lattice structure of relational specification. Though we carry out our discussion for relational specifications, our results can be reinterpreted in any specification model, provided a similar lattice structure is exhibited.

## 3.1 Relational Specifications

We represent the functional specification of programs by relations; without much loss of generality, we consider homogeneous relations, and we denote by $S$ the space on which relations are defined. A relation $R$ on set $S$ is a subset of the Cartesian product $S \times S$. Typically, set $S$ is defined by some variables, say $x, y, z$; whence an element $s$ of $S$ has the structure $s = \langle x, y, z \rangle$. We use the notation $x(s)$, $y(s)$, $z(s)$ (resp. $x(s')$, $y(s')$, $z(s')$) to refer to the $x$-component, $y$-component and $z$-component of $s$ (res. $s'$). We may, for the sake of brevity, write $x$ for $x(s)$ and $x'$ for $x(s')$.Constant relations include the *universal* relation, denoted by $L$, the *identity* relation, denoted by $I$, and the *empty* relation, denoted by $\phi$. Given a predicate $t$, we denote by $I(t)$ the subset of the identity relation

defined as follows: $I(t) = \{(s, s') | s' = s \wedge t(s)\}$. Because relations are sets, we use the usual set theoretic operations between relations. Operations on relations also include the *converse*, denoted by $\widehat{R}$ or $R\widehat{\ }$, and defined by $\widehat{R} = \{(s, s') | (s', s) \in R\}$. The *product* of relations $R$ and $R'$ is the relation denoted by $R \circ R'$ (or $RR'$) and defined by

$$R \circ R' = \{(s, s') | \exists t : (s, t) \in R \wedge (t, s') \in R'\}.$$

The *prerestriction* (resp.*post-restriction*) of relation $R$ to predicate $t$ is the relation $\{(s, s') | t(s) \wedge (s, s') \in R\}$ (resp. $\{(s, s') | (s, s') \in R \wedge t(s')\}$). We admit without proof that the pre-restriction of a relation $R$ to predicate $t$ is $I(t) \circ R$ and the post-restriction of relation $R$ to predicate $t$ is $R \circ I(t)$. The *domain* of relation $R$ is defined as $dom(R) = \{s | \exists s' : (s, s') \in R\}$. The *range* of relation $R$ is denoted by $rng(R)$ and defined as $dom(\widehat{R})$. We say that $R$ is *deterministic* (or that it is a *function*) if and only if $\widehat{R}R \subseteq I$, and we say that $R$ is *total* if and only if $I \subseteq R\widehat{R}$, or equivalently, $RL = L$. Given a relation $R$ on $S$ and an element $s$ in $S$, we let the *image set* of $s$ by $R$ be denoted by $s.R$ and defined by $s.R = \{s' | (s, s') \in R\}$.

### 3.2 Refinement Ordering

We define an ordering relation on relational specifications under the name *refinement ordering*:

**Definition 1** *A relation $R$ is said to* refine *a relation $R'$ if and only if*

$$RL \cap R'L \cap (R \cup R') = R'.$$

In set theoretic terms, this equation means that the domain of $R$ is a superset of (or equal to) the domain of $R'$, and that for elements in the domain of $R'$, the set of images by $R$ is a subset of (or equal to) the set of images by $R'$. This is similar, of course, to refining a pre/postcondition specification by weakening its precondition and/or strengthening its postcondition [7, 10]. We abbreviate this property by $R \sqsupseteq R'$ or $R' \sqsubseteq R$. We admit that, modulo traditional definitions of total correctness [5, 7], the following propositions hold.

- A program $P$ is correct with respect to a specification $R$ if and only if $[P] \sqsupseteq R$, where $[P]$ is the function defined by $P$.
- $R \sqsupseteq R'$ if and only if any program correct with respect to $R$ is correct with respect to $R'$.

Intuitively, $R$ refines $R'$ if and only if $R$ represents a stronger requirement than $R'$. We admit without proof that the refinement relation is a partial ordering (i.e. that it is antisymmetric, reflexive, and transitive).

### 3.3 Refinement Lattice

In [4] Mili et al. analyze the lattice properties of this ordering and find the following results:

- Any two relations $R$ and $R'$ have a greatest lower bound, which we refer to as the *meet*, denote by $\sqcap$, and define by:

$$R \sqcap R' = RL \cap R'L \cap (R \cup R').$$

- Two relations $R$ and $R'$ have a least upper bound if and only if they satisfy the following condition:

$$RL \cap R'L = (R \cap R')L.$$

  Under this condition, their least upper bound is referred to as the *join*, denoted by $\sqcup$, and defined by:

$$R \sqcup R' = \overline{RL} \cap R' \cup \overline{R'L} \cap R \cup (R \cap R').$$

- Two relations $R$ and $R'$ have a least upper bound if and only if they have an upper bound; this property holds in general for lattices, but because the refinement ordering is not a lattice (since the existence of the join is conditional), it bears checking for this ordering specifically.
- The lattice of refinement admits a *universal lower bound*, which is the empty relation.
- The lattice of refinement admits no *universal upper bound*.
- Maximal elements of this lattice are total deterministic relations.

See Figure 3.

In light of this lattice-like structure, we introduce the concept of *orthogonality* between two (sub) specifications, which we had briefly alluded to in section 2.

**Definition 2** *Two specifications $R$ and $R'$ are said to be* orthogonal *if and only if their meet is the universal lower bound of the refinement lattice.*

Consider the analogy with the lattice of the *divides* relation on the set of natural numbers: two natural numbers whose meet (greatest common divisor) is the universal lower bound (1) are said to be *relatively prime*.

If we consider the definition of the *meet* operator in the lattice of refinement, we find that two specifications are orthogonal if and only if their domains are disjoint. By this definition, the specifications $Ord$ and $Prm$ that we discussed in section 2 are not orthogonal. But the decomposition that breaks down the $Sort$ specification into sorting small arrays and sorting large arrays is orthogonal.

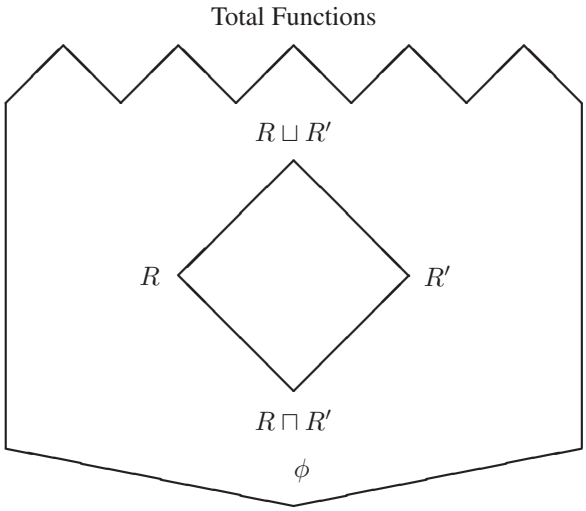


**Figure 3. Lattice Structure of Refinement**

# 4 Failure Costs

## 4.1 Orthogonal Decompositions

We consider a specification $R$ that is decomposed as the join of several component subspecifications, $R_1$, $R_2$, ... $R_k$, and we let $\sigma$ be a system that is intended to satisfy specification $R$. We consider an implicit stakeholder, and we let $\gamma(R_i)$, for $1 \leq i \leq k$, be the failure cost that this stakeholder attaches to subspecification $R_i$. We submit the following definition.

**Definition 3** *If $R_1$, $R_2$, ... $R_k$ are pairwise orthogonal, then the* Mean Failure Cost *of system $\sigma$ with respect to specification $R$ is:*

$$\Gamma(R) = \sum_{i=1}^{k} \gamma(R_i) \times \pi(\overline{S \sqsupseteq R_i}),$$

*where $\pi$ is used to represent the probability of an event, and the overline is used to represent the negation of an event.*

To see why orthogonality is needed, imagine for example a specification that has two components, say $R_1$ and $R_2$, such that $R_1$ refines $R_2$, then the expression

$$\gamma(R_1) \times \pi(\overline{\sigma \sqsupseteq R_1}) + \gamma(R_2) \times \pi(\overline{\sigma \sqsupseteq R_2})$$

counts the failure cost of specification $R_1$ twice, since if $\sigma$ fails to refine $R_2$ then it necessarily fails to refine $R_1$.

More generally, whenever we add up the failure costs of two arbitrary specifications, the cost attached to their meet is counted twice, once for each term. When two specifications are orthogonal, their meet is the universal lower bound, whose failure cost is zero by definition (since, by definition, no specification can fail to refine it).

Since, as we saw earlier, specifications $Ord$ and $Prm$ are not orthogonal, then the formula of definition 3 does not really apply to them. To be accurate, the values we found in section 2 should in fact be considered as upper bounds. Indeed, we generally have the following formula for the mean failure cost,

$$\Gamma(R) \leq \sum_{i=1}^{k} \gamma(R_i) \times \pi(\overline{\sigma \sqsupseteq R_i}),$$

where equality holds whenever the subspecifications are pairwise orthogonal. As an illustration, we consider the specification $Sort$ that we had introduced in section 2, and we consider the following decomposition:

- *Small Sort*, $SS$, is the restriction of relation $Sort$ to arrays that are of size 10 or less.
- *Large Sort*, $LS$, is the restriction of relation $Sort$ to arrays whose size is larger than 10 but no larger than 50.
- *Very Large Sort*, $VS$, is the restriction of relation $Sort$ to arrays whose size is larger than 50.

Let us consider a sixth stakeholder, who we will call *Inspector Searcher*, who searches the output array by in-

spection. This stakeholder does not care whether small arrays are sorted because even if they are not, he can still perform a search on them by hand; As array sizes increase, so does his stake in the correct operation of the sort program. On the other hand, imagine that we have tested the sort routine very thoroughly on arrays of size 1 through 10, and as a result we are very confident that the routine is correct with respect to $SS$. Also, our confidence in its correct behavior decreases with array size. As a result of this hypothetical discussion, we draw the following table, where we record failure probabilities for the various subspecifications and failure costs for the stakeholder at hand, with respect to the various subspecifications.

| Stakeholder | $SS$ | $LS$ | $VS$ |
|---|---|---|---|
| Inspection Searcher | 1 | 10 | 100 |
| Probability of Success | 0.9999 | 0.999 | 0.99 |
| Probability of Failure | 0.0001 | 0.001 | 0.01 |

Clearly, we do have

$$Sort = SS \sqcup LS \sqcup VS.$$

Because relations $SS$, $LS$ and $VS$ have pairwise disjoint domains, they are pairwise orthogonal. Hence we can apply the formula of definition 3 to this decomposition of specification $Sort$. We find: $\Gamma(Sort) = 1.0101$ \$/Hr. This quantity represents the mean failure cost of the proposed system for the cited stakeholder, in light of the proposed failure costs and failure probabilities.

### 4.2 Arbitrary Decompositions

In the section above, we have discussed an example where a specification is decomposed into pairwise orthogonal components, and failure costs are known for each component. In general, however, three conditions prevail:

- Decompositions of the same specification vary from stakeholder to stakeholder.
- Specification components are not pairwise orthogonal.
- Stakeholders do not necessarily know all the failure costs associated with all the components.

Hence we need inference mechanisms for reasoning about failure costs. Defining such an inference mechanism and analyzing it are beyond the scope of this paper; we content ourselves with presenting and discussing some of its rules. But first, we present some examples of failure costs attached to highly overlapping subspecifications. To fix our ideas, we consider a sample/ simplistic example of a flight control system for a passenger airplane, for which we list some sample subspecifications, some sample stakeholders, and some sample failure costs.

Sample Subspecifications:

- Ensure a smooth ride.
- Ensure adherence to flight vector within regulatory tolerance guidelines.
- Ensure timeliness/ adherence to flight schedule.
- Ensure fuel efficiency.
- Ensure adherence to noise pollution standards.
- Ensure timely responsiveness to Auto Pilot setting changes.
- Ensure that the reverse thrust is never applied in mid air.
- Ensure that the landing gears are always activated prior to landing.
- Ensure that the aircraft speed does not fall below the stalling speed for the current flight configuration.

This list is neither complete (of course), nor orthogonal (many requirements overlap a great deal, in the sense that the same functional attributes of candidate implementations will help satisfy them or violate them). As for stakeholders, we consider:

- The airplane's pilot.
- The passengers.
- The Aviation authorities (e.g. FAA).
- The CEO of the airline.
- The CEO of the aircraft manufacturer.
- The CEO of the insurance company ensuring the airline.
- Environmental activists/ organizations.
- Residents in the neighborhood of the airports (departure, destination of the flight).
- The beneficiary of a passenger's life insurance.

Sample Failure Costs:

- CEO of the Airline, Fuel efficiency: Operating Costs.
- CEO of the Airline, Timeliness: Company's reputation for timeliness.

- CEO of the Airline, Smoothness of the flight: Loyalty of the passengers.
- Passenger, Smoothness of the flight: Personal comfort.
- CEO of the aircraft manufacturer, Fuel efficiency: corporate image, selling point.
- Environmental activists, fuel efficiency: carbon imprint.
- Reverse thrust, passenger: Life.
- Reverse thrust, insurance company: Claims.
- Reverse thrust, airline company: Airline safety record.
- Reverse thrust, aircraft manufacturer: Aircraft safety record.
- Reverse thrust, beneficiary of passenger's life insurance: - insurance payout.

# 5 Applications of Mean Failure Costs

In this section we analyze conceptual and practical applications of the metric of *mean failure cost*.

## 5.1 Reliability and Safety

We submit the premise that the mean failure cost encompasses not only reliability, but also safety, in the following sense. Whereas reliability reflects the probability of satisfying the overall system specification, safety focuses on the probability of satisfying specific components of the overall specification, whose failure cost is excessively high. Generally speaking, the system specification, say $R$, refines the safety requirement, say $F$ (see Figure 4). But because the cost of failing $F$ is much higher than the cost of failing $R$, the probability with which we must ensure that we satisfy $F$ must be much higher than the probability of satisfying $R$.

Our metric of mean failure cost does not make a distinction between components with low failure cost and components with high failure cost; rather it deals indiscriminately with a continuum of failure costs. Hence we can imagine a decomposition of $R$ as follows:

$$R = R_1 \sqcup R_2 \sqcup .. \sqcup R_n \sqcup F.$$

From which we infer an upper bound of the mean failure cost as (where we let $R_{n+1}$ represent $F$):

$$\Gamma(R) \leq \sum_{i=1}^{n+1} \pi(\overline{\sigma \sqsupseteq R_i}) \times \gamma(R_i).$$

If we consider the term that corresponds to the safety requirement ($F = R_{n+1}$), we find that since $\gamma(F)$ is very high, the term $\pi(\overline{\sigma \sqsupseteq F})$ must be very low, i.e. $\pi(\sigma \sqsupseteq F)$ must be very high. In other words, we must ensure, with the greatest level of confidence, that $\sigma$ refines the safety requirement.

Figure 4 illustrates how we can fold safety into the calculation of reliability by the mean failure cost. The left figure (a) shows the structure of the overall specification ($R$) as the join of individual subspecifications, and highlights that the overall specification refines (most typically) the safety requirement. The right hand figure (b) shows schematically the set of candidate implementations that meet the reliability requirement (triangle rooted at $R$) and the set of candidate implementations that meet the safety requirement (triangle rooted at $F$). The space between the two triangles (higher than $F$ but no higher than $R$) is the set of systems that are safe but not reliable. Even though logically, any system that refines (is higher than) $R$ refines $F$, this does not mean that a reliable system is necessarily safe, as we usually consider a system to be safe if it can be assured to refine $F$ with a much higher probability than the probability of refining $R$.

## 5.2 The ROI of V& V

We consider a system $\Sigma$ which is being operated by a set of stakeholders $K_1$, $K_2$, ... $K_n$. We envisage to take some V& V action, and we want to answer two questions:

- How can we tell whether the V& V action will pay off?
- Assuming that we find it to pay off, how can we distribute its cost on the stakeholders?

We submit an ROI based solution to these questions, using the metric of mean failure cost. An ROI is typically defined by the following parameters:

- A cycle length (counted in years), that we denote by $Y$.
- A discount rate (abstract number), which represents the time value of money, that we denote by $d$ (1 dollar today is worth $(1 + d)$ dollars next year).
- A cost function $C(y)$, that maps years from 0 to $Y$ onto costs (valued in dollars).
- A benefit function $B(y)$, that maps years from 0 to $Y$ onto benefits (valued in dollars).

With these parameters, the $ROI$ is computed according to the following formula (where we assume $B(0) = 0$):

$$ROI = \frac{1}{C(0)} \times \sum_{y=0}^{Y} \frac{B(y) - C(y)}{(1+d)^y}.$$

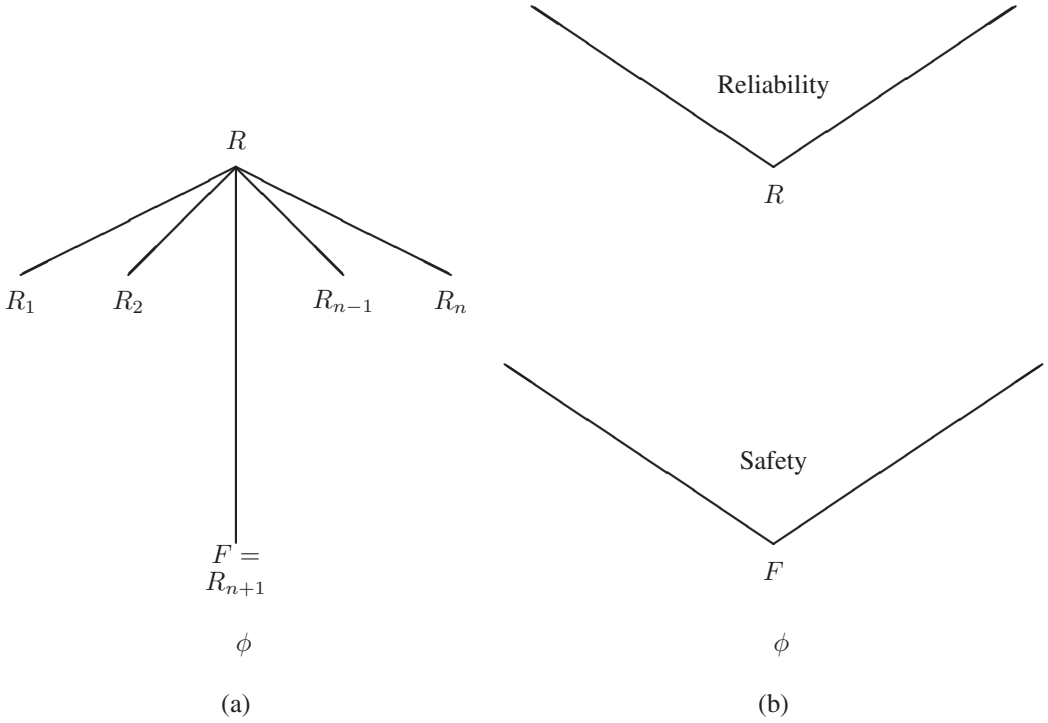


**Figure 4. Integrating Reliability and Safety**

The approach that we propose can be characterized by the following premises:

- The decision of whether the V& V action is worthwhile can be made separately for each stakeholder.
- Each stakeholder computes his own ROI based on his contribution to the cost of V& V action, and the benefits he gains in terms of reduced mean failure cost.
- The cost of the V& V action must be distributed in such a way as to make the ROI positive for all stakeholders.
- One posible formula for distributing the cost of the V& V action is to prorate it to the reduction in MFC for each stakeholder.

Quantitatively, this can be modeled as follows:

- $Y_i$ is the investment cycle of stakeholder $K_i$; there is no reason to assume that all stakeholders have the same cycle.
- $d_i$ is the discount rate for stakeholder $K_i$; there is no reason to assume that all stakeholders have the same discount rate.
- $C_i(0)$ is the portion of the cost of the V& V action that was billed to stakeholder $K_i$. If we let $\Gamma$ be the cost of the V& V action, then this cost must be borne by the stakeholders, i.e. $\Gamma = \sum_{i=1}^{n} C_i(0)$.
- $B_i(y)$ is the reduction in mean failure cost at year $y$ for stakeholder $K_i$. This is computed for each stakeholder as the difference between the MFC before and after the V& V action.
- For simplicity we assume $B_i(0) = 0$ and $C_i(y) = 0$ for all $y$ between 1 and $Y_i$.
- Then we define the return on investment of stakeholder $K_i$ as:

$$ROI_i = \frac{1}{C_i(0)} \times \sum_{y=0}^{Y_i} \frac{B_i(y) - C_i(y)}{(1+d_i)^y}.$$

- Then the V& V action is deemed worthwhile if the $ROI_i$'s are positive for all stakeholders.

As an illustration, we consider the simple example we had presented earlier and consider the following V& V scenario: We are interested in augmenting our sort routine with a fault tolerant capability that detects errors in the execution trace and invokes a recovery routine when errors are detected. To minimize space overhead and complexity, we resolve that the error detection assertions deal

with the array being ordered, but no with the array being a permutation of the initial array; hence this fault tolerant capability alters the probability of failure with respect to $Ord$, but leaves the probability of failure with respect to $Prm$ unchanged. Specifically, we find that the deployment of this measure increases the probability of satisfying $Ord$ from 0.99 to 0.994; furthermore, we estimate that the deployment of this measure (in terms of manpower) costs 400 $. Table 5 shows some details in the calculation of the ROI's for the five stakeholders; the yearly benefit is calculated for each stakeholder from the MFC by considering the number of hours of operation that each stakeholder peruses. The investment cost ($C_i(0)$) billed to each stakeholder is prorated to the yearly benefit of the stakeholder; the sum of the bills does add up to the cost of the V& V operation. As a stakeholder-independent measure of whether the V& V action is a worthwhile investment, we can compute its overall NPV as the sum of stakeholder NPV's (the ROI's are not additive, but the NPV's are); this is given in Figure 5.

# 6 Back to Security

In the foregoing discussion, we have explored how reliability and safety can be quantitatively modelled in a way that, we feel, is more meaningful/ useful than the traditional MTTF. We argue that the same idea can be applied to security; the mean failure cost (per unit of time) would then measure the mean loss that a stakeholder assesses as a result of possible security breaches. This function would depend on the likelihood of a security breach as well as the impact that a breach may have on the stakeholder. The likelihood of security breaches would, in turn, depend on vulnerabilities, security counter measures, perpetrator behavior, intrusion protections, insider threats, etc.

A critical feature of our approach is its dependence on the lattice of refinement, and the structure that this lattice confers complex system specifications. In order for this approach to be applied to security, a similar refinement structure must be built for security attributes. In [9] Mili et al. cast security attributes in terms of refinement calculi, by

- Modelling security requirements as behavioral attributes that focus on effects (enforcing security policies) rather than causes (absence of vulnerabilities).

- Modeling security attributes of a system as externally observable system behaviours rather than possible remedies (e.g. firewalls, access controls, authentication protocols).

- Certifying the security properties of a system by confronting verifiable security attributes of candidate systems against prespecified security requirements.

The next step is to generalize this model towards a refinement calculus similar to those that were developed for producing correct programs or for verifying program correctness [6–8, 10, 13].

# 7 Conclusion

## 7.1 Summary

In this paper, we propose a tentative measure of reliability that offers the following advantages over the traditional MTTF:

- It reflects the variance that may exist amongst different stakeholders of the same system.

- For a given stakeholder, it reflects the variance that may exist amongst the stakes she/he attaches to meeting each requirement.

- For a given compound specification, it reflects the variance that may exist amongst the levels of verification and validation that are performed on components of the specification.

## 7.2 Assessment

This work is clearly in its infancy; this paper merely presents the idea of measuring reliability (and safety) with mean failure cost, and discusses some of the mathematics that must be explored to make this metric practical. Our first impressions of this measure are positive, in light of the following premises:

- *The proposed metric is consistent with the spirit of Value Based Software Engineering* [1–3]. Whereas the MTTF is an abstract quantity that reflects the failure rate of a system, the MFC quantifies the impact of failures by providing a failure cost per unit of time. This cost must be balanced against the benefit of operating the system for the same unit of time, to determine the desirability of operating the system.

- *The proposed metric integrates reliability and safety in a unified model*. The mean failure cost makes no distinction between reliability and safety, as it supports a continuum of failure costs, from the mundane to the critical, hence it integrates safety seamlessly with reliability.

| Stakeholders | $Ord$ | $Prm$ | MFC Before | MFC After | MFC Gain | Hrs of Oper. | $B_i(y)$ | $C_i(0)$ | $Y_i$ | $d_i$ | $ROI_i$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Bin. Searcher | 10 | 2 | 0.1002 | 0.0602 | 0.0400 | 2000 | 80 $ | 115.20 $ | 3 | 0.15 | 0.78 |
| Median Finder | 4 | 4 | 0.0404 | 0.0244 | 0.0160 | 3000 | 48 $ | 69.00 $ | 3 | 0.12 | 0.67 |
| Avg Finder | 1 | 3 | 0.0103 | 0.0063 | 0.0040 | 2500 | 10 $ | 14.40 $ | 2 | 0.05 | 0.29 |
| Lin. Searcher | 7 | 10 | 0.0710 | 0.0430 | 0.0280 | 5000 | 140 $ | 201.40 $ | 2 | 0.10 | 0.21 |
| B. F. Searcher | 0 | 10 | 0.0010 | 0.0010 | 0.0000 | 500 | 0.0 $ | 0.00 $ | 4 | 0.15 | 0.00 |
| P(Fail) Before | 0.010 | 0.0001 | | | | | | | | | |
| After | 0.006 | 0.0001 | | | | | | | | | |
| Global Cost | | | | | | | | 400.00 $ | | | |
| Global NPV | | | | | | | | | | | 582.56 $ |

**Figure 5. Calculating the ROI of a V& V Action**

- *The proposed metric supports value based decision making*. Traditionally, verification and validation effort is charged uniformly on all stakeholders. With the quantification infrastructure that we have introduced to compute mean failure cost, we can imagine a scheme where the cost of any V&V effort is charged on the stakeholders according to what they stand to gain from it: Hence if a particular V&V effort is aimed at improving the level of confidence that $\sigma$ refines a component $R_i$, then stakeholders are charged according to the stake they have in satisfying $R_i$. We are also considering to integrate *verification costs* into the mix, to take into consideration the fact that it may be easier to verify a system against one specification component than against another. Such costs depend on the system $\sigma$, the requirement $R_i$, and the selected verification method.

### 7.3 Prospects

On the practical side, we need to find sample applications where deployment of the MFC metric shows its usefulness / superiority, by providing a sound basis for analysis and decision making. On the theoretical side, we need to develop the mathematical infrastructure that allows us to estimate or to approximate the MFC using failure costs and failure probabilities given (respectively) by stakeholders and engineers (V&V teams).

## References


[1] S. Biffl, A. Aurum, B. W. Boehm, H. Erdogmus, and P. Gruenbacher. *Value Based Software Engineering*. Springer Verlag, 2006.

[2] B. W. Boehm and L. Huang. Value based software engineering: A case study. *IEEE Computer*, 36(3), march 2003.

[3] B. W. Boehm and L. Huang. Value based software engineering: Reinventing earned value monitoring and control. *ACM Software Engineering Notes*, 28(2), march 2003.

[4] N. Boudriga, F. Elloumi, and A. Mili. The lattice of specifications: Applications to a specification methodology. *Formal Aspects of Computing*, 4:544–571, 1992.

[5] E. Dijkstra. *A Discipline of Programming*. Prentice Hall, 1976.

[6] P. Gardiner and C. Morgan. Data refinement of predicate transformers. *Theoretical Computer Science*, 87:143–162, 1991.

[7] D. Gries. *The Science of programming*. Springer Verlag, 1981.

[8] E. Hehner. *A Practical Theory of Programming*. Springer-Verlag, 1993.

[9] A. Mili, A. Vinokurov, L. L. Jilani, F. T. Sheldon, A. Thomasian, and R. B. Ayed. Modeling security as a dependability attribute: A refinement based approach. *Innovations in Systems and Software Engineering, A NASA Journal*, 2(1):39–48, 2006.

[10] C. Morgan. *Programming from Specifications*. International Series in Computer Sciences. Prentice Hall, London, UK, 1998.

[11] D. Siewioreck and R. Swarz. *Reliable Computer Systems, Design and Implementation*. Digital Press, 1992.

[12] F. Stevens, T. Courtney, S. Singh, A. Agbaria, J. F. Meyer, W. H. Sanders, and P. Pal. Model based validation of an intrusion tolerant information system. In *Proceedings, SRDS*, pages 184–194, 2004.

[13] J. V. Wright. A lattice theoretical basis for program refinement. Technical report, Dept. of Computer Science, Åbo Akademi, Finland, 1990.