Ranking Weakness Findings

Dr. Nikolai Mansourov, CISSP
KDM Analytics

Abstract
This white paper describes the roadmap towards scoring systems for weakness findings within the context of the Static Code Analysis (SCA) approach. Methods of expert-based scoring have been well-developed and adopted in the vulnerability management industry, even though some of the proposed measurements are quite subjective. In the context of automated SCA tools the design of a scoring system has significant challenges because score calculations must be done automatically and score values from multiple SCA tools must be merged. Various SCA tools use proprietary, not well-documented rankings, involving different scales, a mix of bounded and unbounded scores, and scores with widely different factors. While the upcoming standards like the OMG TOIF and the OASIS SARIF facilitate the agreement on weakness identities reported by multiple SCA tools, normalizing the scores without relying on manual inputs remains a challenge which prevents development of common risk analytics.

This white paper frames the discussion of scoring systems for SCA, provides a survey of existing approaches and outlines the requirements for a weakness scoring system, with the emphasis on the role of standards and information exchange in this area.

In the long term, a common scoring system should emerge and standardization efforts should be focused at designing and promoting standard measurements for weakness findings that provide the foundations for the common scores. However, in the short and medium term, the options are limited to 1) providing a framework to take proprietary measurement results as input and normalize them, and 2) designing a common post-merge scoring system that does not rely on the measurement results provided by individual tools (for example, when there is none available), but uses other characteristics of weakness reports instead.

DISCLAIMER: the views expressed in this white paper do not necessarily reflect the views of the Object Management Group (OMG), or the OMG System Assurance Task Force, or any of the member organizations of the OMG.

1. Introduction

How is a weakness finding turned into data that can support decision-making?

Weakness finding is a claim that a potential issue exists in the code under investigation. The term “weakness” means that some of the computations implemented by the code may cause failures, perhaps under special circumstances created by an intentional attack. By a broader definition, a weakness is any place in the code that one has decided to flag according to some rule (for example, a corporate style guide, use of a questionable construct, a missing semicolon, a typo). Weakness findings are often produced automatically by Static Code Analysis (SCA) tools that scan the source or machine code of an entire software system or an individual module and analyze control and data flows in search for the indicators of potential weaknesses. The idea of using rules to automatically flag certain places in code and produce lists of findings is central to the SCA approach.
Ranking Weakness Findings

Two characteristics are essential to a weakness finding: the type of a weakness (related to the rule that was used to flag the issue) and the location of the weakness in the code.

A weakness corresponds to a certain place in the code. As such, the weakness is reported at a certain location of the code – for example, a line of code - which means that only the computations that involve this location may cause failures. In the SCA approach, a computation is a manifestation of a control and data flow through the code, or, in simpler situations, simply a place in the code that was flagged by a certain rule.

Weakness type is related to a rule that was used to identify the finding in the code. Weakness type describes a failure, or as frequently done by the quality assessment tools, describes what is wrong with the code. A weakness is usually described as one of the several known weakness types – a vocabulary of possible situations. Common Weakness Enumeration (CWE) is an example of such a vocabulary [1]. An SCA tool usually uses a proprietary collection of weakness types (rules) that it scans for.

Assigning a rank to each weakness finding allows prioritization of weakness findings and thus supports weakness remediation decisions. Some types of weaknesses may cause more impact than others and it makes sense that more impactful weakness findings are investigated and fixed earlier than the less impactful ones. This prioritization is particularly relevant for the developers of the code, who presumably want to make the code as secure as possible by fixing all known issues, regardless of the context in which the code will be used (the context cannot be assumed to be always known, especially in the case of a software library). Weakness ranking gives a useful means to manage development resources, possibly together with the means to better categorize, filter and group the weakness findings.

“Software developers often face hundreds or thousands of individual bug reports for weaknesses that are discovered in their code. In certain circumstances, a software weakness can even lead to an exploitable vulnerability. Due to this high volume of reported weaknesses, stakeholders are often forced to prioritize which issues they should investigate and fix first, often using incomplete information. In short, people need to be able to reason and communicate about the relative importance of different weaknesses.” [2]

More formally, a ranking is a relationship between a set of items such that, for any two items, the first is either 'ranked higher than', 'ranked lower than' or 'ranked equal to' the second [3]. A rank is a relative position of an item in a set with a ranking. In other words, ranking defines an order of items in a set. So, there are three components to a “rank”: 1) the method - the “ranking”; 2) the scope – the set of items to which ranking applied, and 3) the rank of a given item – an observation resulting from applying the “ranking” to the items in the scope.

These considerations allow multiple ways in which a rank can be associated with a weakness finding. As the first example, let’s consider a ranking where the scope is the set of weakness findings produced by one SCA tool for the given code base. For example, given a set of weakness findings, a rank may represent “importance” of a finding within this set. Given a set of 98 findings, each finding is assigned a unique rank number between 1 and 98. Given a set of 201 findings, the rank will be between 1 and 201.
As the second example, let’s consider a ranking where the scope is the set of all weakness types (regardless of the system in which they were identified), so that all findings of the same weakness type will be ranked as “equal”. Now, given a set of findings by one SCA tool for a given code base, the second ranking method will produce different ranks of the findings than the first example. The second order represents absolute importance of a finding (based on its weakness type) either for a given organization, or broadly, across multiple organizations within the same industry – but not the relative importance of a finding within a given code base. The value of the rank for a set of 98 findings will still be between 1 and 98. Later in the paper, we will briefly describe the CWRAF system [4] that allows building customized “top-N-lists” for an industry, and is used to define rankings.

The second ranking example shows that it may be interesting to see the actual values in addition to the rank. In statistics, "ranking" often refers to the data transformation in which some numerical or ordinal values – often called “scores” - are replaced by their rank when the data are sorted. For example, the numerical data 3.4, 5.1, 2.6, 7.3 are observed, the ranks of these data items would be 2, 3, 1 and 4 respectively. For example, the ordinal data hot, cold, warm would be replaced by 3, 1, 2. In these examples, the ranks are assigned to values in ascending order. In some other cases, descending ranks are used.

The second example also demonstrates the importance of the “granularity” of a score – how many individual items in a set will have the same score – since using a score with a coarse granularity many items are likely to be scored equally, and therefore their rank will be not meaningful. Items with the same score will be ranked in an arbitrary way.

A score often implies an objective measurement approach, where the score is calculated based on specific contributing factors using a repeatable method based on objective criteria. Given a repeatable score, a set of items can be sorted by their scores, producing a rank. Such rank will be more objective. However, items can be ranked without defining a score. We have already provided two examples which have defined a ranking, but did not reveal the ranking method.

The method for calculating a score in the above examples may seem rather subjective and ill-defined. A more objective approach to associate a number with a weakness finding is to define the score as the “criticality” of a finding in terms of its potential impact to the system or the organization. Structured multi-dimensional scores like CWSS take many factors into account [2].

Lack of a good scoring system for weakness findings in the context of an SCA approach is a major gap that prevents development of powerful software assurance tools, and risk analytics.

The goals of this paper are to frame the discussion of the scoring systems within the context of the SCA approach, provide the survey of existing approaches and to define the requirements for a good weakness scoring system, with the emphasis on the role of standards and information exchange in this area.
2. Related efforts

Ranking and scoring was initially explored in the vulnerability management industry – patching commercial software to remediate their known vulnerabilities – and standardized as the NIST SCAP [5,6] ecosystem.

2.1. The challenges of managing vulnerabilities

The term “vulnerability” refers to a confirmed weakness claim that involves a known commercially-available software package.

“The challenge security teams face managing and remediating security defects has evolved in recent years. Basic vulnerability scanning is no longer the challenge. Organizations face a number of issues as part of their vulnerability management programs, not the least of which is data management. Mature security teams are assessing risk across all of their asset layers including their applications, databases, hosts and networks. Any group dealing with a sizable environment isn’t struggling with finding security defects, but rather with managing the mountain of data produced by their vulnerability assessments, penetration testing, and threat modeling in order to fix what’s most important first” [7].

According to [7], “security is now a data problem”. Do you have a data scientist on your security team?

2.2. NIST SCAP

Since most of the enterprises use known commercially-available operating systems and software packages, analysis of the vulnerabilities for an enterprise can be decoupled into two phases: analyzing vulnerabilities in commercial software and analyzing systems that use this software. This approach is practical as an ecosystem of automated tools can be built around the information flow from vulnerability researchers to a vulnerability repository and further - to the tools that scan a host to identify the known commercial software packages. Scanning tools can then access a repository of known vulnerabilities and report the list of vulnerabilities relevant to the host.

NIST Security Content Automation Protocol (SCAP) [5] standardizes the format and nomenclature by which software vulnerability and security configuration information is communicated, both to machines and humans.

The “NIST SCAP vulnerability” claims usually involve only an indirect reference to the computations affected. Vulnerability claims are reported using a variety of methods, often by an accidental observation of a failure, or by observing malware behavior and thus gaining insight into the weakness being exploited, or through deliberate dynamic testing. Some vulnerabilities are reported by the vendors who have better understanding of the code. This means that vulnerability reports take the “black box” of the software package. Reports of vulnerabilities in known commercially-available software are collected in the so-called vulnerability repositories, such as the National Vulnerability Database (NVD) [8,6].
Ranking Weakness Findings

Remediation of vulnerabilities in commercial packages is limited to applying patches from the vendors. Therefore, vulnerabilities described in the NVD are linked to patch versions and to the means that can be used to identify the presence of the corresponding software package, such as filenames and registry entries.

“NIST SCAP vulnerabilities” are not linked to locations in the software code. In contrast, the SCA weakness findings are linked to control and data flow of the code, allow remediation by fixing the issue (producing a patch for deployed systems), making the SCA approach available for newly developed software, proprietary systems (one-off deployment, limited possibility to observe and report vulnerabilities by third-parties), and more systematic, not relying on the content in a vulnerability database, but rather analyzing the code from the first principles. As the result, the SCA weakness findings are often more ambiguous regarding the possible impact or cause, since they are analyzed using an incomplete context [6].

Over the years, various methods for prioritization vulnerabilities have developed, all with advantages and disadvantages for different organizations.

2.3. Simple bounded scoring
The first method to emerge was a simple ranking of results along the lines of High, Medium and Low. Of course, you can imagine that other labels might be applied, but fundamentally, you can see from any ranking which vulnerabilities should be fixed first.

Very quickly, however, information security professionals started wondering what to do when the number of High vulnerabilities exceeded the time and scope available. One solution that seems like a good idea is to make the ranking more granular, say 1-5 or 1-10.

This, of course, doesn’t solve the problem, but simply makes it more manageable for a time. You still, ultimately, can’t distinguish between two vulnerabilities with the same ranking. And because the ranking is subjective, you can’t decompose it to any rational components.

Simple ranking uses a linear scale or ordinal values. This method is subjective when applied to measurements of complex characteristics. A score, on the other hand, is calculated based on specific underlying criteria with a repeatable method, that implies a multi-dimensional rather than a linear scale. Calculating of a score is based on multiple measures to each of the contributing factors. A multi-dimensional value is then transformed to the final ordinal value on a linear scale. As a measurement method, scoring represents a metric that is based on multiple measures, each more elementary and therefore more objective.

2.4. Tripwire unbounded score
In order to address the “clustering” issue, some organizations support unbounded scores. nCircle, now a part of Tripwire, started producing an unbounded vulnerability score, which frees it from the risk of clustering conditions at the top, i.e. too many ‘Highs’. In order to accomplish this, the Tripwire score is generated from three criteria: the depth of access the vulnerability provides if exploited (risk), the external exploit characteristics (skill), and the time that has elapsed since the vulnerability was published (time) [9].
2.5. CVSS bounded score
NIST SCAP involves a scoring system for vulnerability, called Common Vulnerability Scoring System (CVSS) [10]. Common Vulnerability Scoring System (CVSS) is an example of a bounded score.

CVSS was conceived as a means to score vulnerabilities in the world at large, rather than an instance of a vulnerability on a host in your network. For that reason, CVSS decreases the vulnerability score when a patch is available. In general, a vulnerability presents less risk once it can be patched, but for a given unpatched host the risk cannot be decreased, while for a patched host the vulnerability may not have any risk at all.

CVSS also takes the approach of a bounded score, ranging from 0 to 10, with 10 being the most severe. Scores are calculated based on a formula that depends on several metrics that approximate ease of exploit and the impact of exploit. As the result, CVSS suffers from the clustering problem mentioned above. You can still, and frequently may, end up with more 10s than you can reasonably address.

One of CVSS' strengths lies in its simplicity. CVSS divides the overall score into 14 separate characteristics within three metric groups: Base, Temporal, and Environmental. Each characteristic is decomposed into two or more distinct values. For example, the Access Vector reflects the location from which an attacker must exploit a vulnerability, with possible values of Local (authenticated to the local system), Remote (across the network), or Network Adjacent (on the same physical or logical network). Typically, in addition to the CVSS score, a vector is provided that identifies the selected values for each characteristic.

With the associated documentation, CVSS scoring is fairly repeatable, i.e., different analysts will typically generate the same score for a vulnerability. While different scores can be generated when information is incomplete, and significant variation is possible if an analyst does not closely follow documentation, CVSS does provide the consistency that is useful for non-expert system and network administrators for prioritizing vulnerabilities.

CVSS has been widely adopted, especially the use of base scores from the Base metric group. Some organizations use the Temporal and Environmental portions of CVSS, but this is relatively rare, so these metric groups may not have been sufficiently vetted in the real world.

2.6. Ranking, scoring and categorization
Scoring and ranking can be augmented in extremely valuable ways through categorization [9]. Broadly, categorization simply means applying a label to the conditions in some systematic way. This includes labels that are simply semantically valuable, as well as those that directly affect a score calculation.

For example, you might categorize a vulnerability by its impact on the target system; does it impact confidentiality, integrity or availability. In fact, CVSS does just that, and uses that as a criterion in its calculation. Another valid category might be vulnerabilities that affect web servers or encryption.
These are not generally used in calculating scores, but they may be valuable for other purposes, like filtering results for specific groups. Filtering and categorization may be a useful part of the prioritization process.

3. Standard Terminology

There is some confusion related to the terms score, measurement, measure and metric. So let’s review the standard terminology.

3.1. Scores

Merriam-Webster dictionary defines score as a number that expresses accomplishment (as in a game or test) or excellence (as in quality) either absolutely in points gained or by comparison to a standard. Score is often used as a basis for ranking. A score is a particular form of a measure. For example, a FICO score is a way of measuring an individual's creditworthiness. A FICO score is a quantification of a variety of factors in an individual's background, including a history of default, the current amount of debt, and the length of time that the individual has made purchases on credit. A FICO score ranges between 350 and 850. In general, a score of 650 is considered a "fair" credit score, while 750 or higher is considered "excellent." A FICO score is a convenient way to summarize an individual's credit history and is included in a credit report. The term comes from the Fair Isaac Corporation, which created the system.

3.2. Measures and Metrics

“The term metric is often used to refer to the measurement of performance, but it is clearer to define metrics and measures separately. A measure is a concrete, objective attribute, such as the percentage of systems within an organization that are fully patched, the length of time between the release of a patch and its installation on a system, or the level of access to a system that a vulnerability in the system could provide. A metric is a useful and abstract (somewhat subjective) attribute, such as how well an organization’s systems are secured against external threats or how effective is the organization’s incident response team. An analyst can approximate the value of a metric by collecting and analyzing groups of measures.

Historically, many measurement efforts have focused on collecting individual measures, and given little or no thought as to how those measures could be combined into useful metrics. Ideally, organizations should first select their metrics, and then determine what measures they can perform that support those metrics. An organization should can have multiple levels of metrics, each geared toward a particular type of audience. For example, technical security staff might be interested in lower-level metrics related to the effectiveness of particular types of security controls, such as malicious code detection capabilities. Security management might be interested in higher-level metrics regarding the organization’s security posture, such as the overall effectiveness of the organization’s incident prevention and handling capabilities. Lower-level metrics facilitate making more tactical decisions, whereas higher-level metrics are well suited for making more strategic decisions. The lower-level metrics are often used as input to the higher-level metrics.” [19].
3.3. Measure and measurement
Standard terminology related to measurement is defined in the International Vocabulary of Metrology (VIM) [11]. Section 2 gives the following definitions.

**Measurement** - process of experimentally obtaining one or more quantity values that can reasonably be attributed to a quantity. Measurement does not apply to nominal properties.

**Measurement principle** - phenomenon serving as a basis of a measurement

**Measurand** - quantity intended to be measured. The specification of a measurand requires knowledge of the kind of quantity, description of the state of the phenomenon, body, or substance carrying the quantity, including any relevant component involved. In the second edition of the VIM and in IEC 60050-300:2001, the measurand is defined as the “particular quantity subject to measurement”.

**Measurement method** - generic description of a logical organization of operations used in a measurement

**Measurement procedure** - detailed description of a measurement according to one or more measurement principles and to a given measurement method, based on a measurement model and including any calculation to obtain a measurement result

**Measurement result** - set of quantity values being attributed to a measurand together with any other available relevant information

**Measurement model** - mathematical relation among all quantities known to be involved in a measurement.

3.4. Measurement Scales
When we talk about metrics and measurements, we usually think about those for mass (gram) and time (second). We can add, compare, and average values. But other scales can be useful.

In the taxonomy proposed by Stevens, the most basic "scale" is nominal, which is just a classification. VIM calls them "nominal properties" (1.30). Some nominal properties are gender, feelings (happy, serious, angry), language (C, Java, Ada), and ISO two-letter country code. Nominal properties may be hierarchical. For example, Italic, Celtic, and Germanic are all Indo-European languages, and computer instruction sets are RISC or CISC.

An ordinal scale is a nominal scale that has a natural order. For instance, weather reports give hurricane intensity on the Saffir-Simpson hurricane scale a discretization of wind speed. The Mohs scale grades relative hardness from 1 (talc) to 10 (diamond). The scale is not uniform: diamond (hardness 10) is 4 times harder than corundum (hardness 9) and 6 times harder than topaz (hardness 8).

An interval scale has equal distance between values. Values on an interval scale cannot be added: it makes no sense to ask what is the sum of 4 July 1776 and 23 December 1805. The number of days between them (difference) is reasonable (10 763 days).
A ratio scale is an interval scale with a zero. 20 K (degrees Kelvin) is twice as hot as 10 K. (20°C is not twice as hot as 10°C; Celsius is an interval scale.)

Some scales naturally report in logarithmic terms. The Richter scale was a logarithmic scale for earthquakes, which measured the size of earthquake waves, not an earthquake's total energy. Relative sound energy is often measured with a logarithmic unit, the bel.

Linear scales may not always fit well. Color is better described in three-dimensional models, such as hue, value, and chroma in the Munsell color model.

3.5. Observations
The term “measurement” is often used ambiguously as either the “process of obtaining values” (standard usage) or as the “measurement result” (the values obtained as the result of applying measurement to a particular subject). Performing a given measurement for a given set of items results in measurement results, or observations. Observations are the specific quantity values associated with the set of items. An observation is a report, and as such it always implies provenance characteristics: when measurement was performed, what items were measured, who performed the measurement, etc.

3.6. Bounded and Unbounded Scores
The main purpose of a score is to compare items in some collection, usually based on some abstract quantity. Selection of a measurement scale determines the possible kinds of comparisons. For example, a salary of $1,000,000 is 10 times higher than a salary of $100,000, or a High confidence is higher than a Medium confidence.

Some measures are fundamentally unbounded. For example, salary is unbounded. The intensity of light is unbounded.

Since a scoring system is a measure designed to support decision-making, bounded measures are usually preferred. This happens primarily because the repertoire of different decisions tends to be a finite (and small) enumeration, so certain ranges of a score can be mapped to different decisions. When pushed to the extreme, a nominal scale is ideally suited for supporting decision-making. For example, Red-means stop, Green-means go.

From the decision-making perspective, an unbounded measure can be transformed into a bounded one. Such transformation is often referred to as normalization. A very common trick to do so is to use the hyperbolic tangent tanh as the 'squashing function'. It automatically fits all numbers into the interval between -1 and 1. Another squashing function is the logistic function defined as $f(x) = 1/(1 + e^{-x})$, which restricts the range from 0 to 1 (with 0 mapped to .5).

Statistics usually deals with a collection of observations, with is bounded. This allows the use of means, standard deviations, quintile, decile, percentile, standard z-scores, etc. even if the value in question is unbounded. Understanding the mean allows defining a standard z-score. Z-score is the signed number of standard deviations by which the value of an observation or data point is above the mean value of what is being observed or measured. Observed values above the mean have positive standard scores, while values below the mean have negative standard scores. The standard score is a dimensionless quantity obtained by subtracting the population mean from an
individual raw score and then dividing the difference by the population standard deviation. This conversion process is called standardizing or normalizing.

Using unbounded measure for decision-making may lead to the need to redesign the score, for example if the mean of an unbounded value changes over time. A score based on an unbounded value may need to be periodically adjusted, which complicates the use of this score for historical comparisons over larger periods of time.

3.6 Static (fixed) and dynamic (sliding) scales for ranking

Ranking with a static (fixed) scale is suitable for comparisons across large populations of items (and possibly performing historical comparisons and trend analysis over large spans of time). For example, we want to measure complexity of a weakness finding as a number of code paths that supply data values to the reported location. As the number of code paths is an unbounded value, we will use a logarithmic scale as follows: 0 for 0-9 paths, 1 for 10-99 paths, 2 for 100-999 paths, 3 for 1000-10,000 paths, and so on. For practical purposes, we can consider a bounded scale 0-10. This scale allows us to discern all possible situations. As the tools improve and tackle weaknesses with more complexity, we would be able to observe this using our fixed scale as the increase in some kind of an industry averages, when we will manage to collect the data.

However, if we want to apply this scale for a small number of findings, - usually those resulting from the same system under assessment, we may find ourselves in a situation, where all findings would fall into the same range, and the ranking would not be discerning. For example, when we have 5 findings with code paths as follows: 0,0,5,8,9 they will all be ranked as 0. To address this situation, a dynamic (sliding) scale may used such that a rescale operation is applied to adjust the scale and make the rakings more discerning. For example, we can adjust to a scale 0 for 0,1 paths, 1 for 2,3 paths, 2 for 4,5 paths, 3 for 6,7 paths, 4 for 8,9 paths. Then the findings will be ranked as 0,0,2,4,4 which is more discerning. After the adjustment, the new scale provides a better local ranking, but the rank values cannot be used in a larger context, e.g. for comparing this system under assessment to other systems within the same industry. Note, that using the raw unbounded measure of the number of code paths per finding is discerning in both cases, but does not support decisions very well, as it must be transformed into a grade measure with a limited number of choices that are commensurate to the repertoire of the decisions we intend to support.

4. The Challenges of a Weakness Scoring for the SCA approach

What are the challenges to designing a weakness measurement system that can support weakness prioritization decisions for a single system under investigation in the context of the SCA approach? Based on the previous discussion, we are looking at designing a score – a measurement method that is based on multiple individual measures to each of the contributing factors, using a multi-dimensional scale. Individual factors may be also useful for the categorization and filtering of the weakness findings. The multi-dimensional value is then transformed to the final ordinal value on a linear scale and used to calculate the rank of each finding. As a measurement method, scoring represents a metric that is based on multiple measures, and as long as each measurand is rather elementary, the individual measures are objective and the overall measurement method repeatable.
There are several challenges with designing a good score for weakness findings. Some challenges are similar to the scores for vulnerability management. They include selection of the contributing factors, measurement scales, individual measurement methods and the linearization formula that exhibits desired mathematical characteristics, such as the expected distribution and clustering.

4.1. Scores for SCA
Several characteristics of the SCA approach make it different from vulnerability management, and this introduces additional challenges to the design of a good score. These characteristics are related to the fact that SCA performs analysis of control and data flow of code. This process is complex, and in general not tractable, which means that some interesting weaknesses may not be fully recognized or validated. This leads to a situation where an SCA tool can produce a false report. Designers of an SCA tool need to strike a balance between the cost of a deep analysis and the usefulness of the potential finding. The promise of the SCA approach is the coverage. Another challenge is a possibly insufficient context for the analysis, which leads to the situation where either the full control and data flow is not known to the SCA tool, or the consequences of a weakness are not known. In contrast, vulnerability reporting is an opportunistic process: when a vulnerability is reported against a known commercial product, it has already been confirmed, and the impacts are well understood. The disadvantage of (manual) vulnerability reporting is its unknown and insufficient coverage.

Assigning a score to each weakness may be done based on the weakness type alone or by considering the characteristics of the individual weakness finding. We believe, that there are several aspects of an individual finding that contribute to a score:

- Confidence in the finding: based on internal technology aspects, has this finding been fully validated, or were some approximations used?
- Architectural aspects: which computations will cause failures? how important they are? how many of them? how can an attacker control the computation and inject data?
- Impact aspects: what are the impacts of the failure?

4.2. Designing scores for automated calculation
One of the major challenges for designing a score in the context of SCA tool is that the measurement must be produced by an automated capability (an SCA tool). Designing a score for automation implies an objective and repeatable measurement process. As we are demonstrating some of the existing approaches, many scores are designed for the use by human experts. When designing a score for automation, discernable objective measures should be preferred, and opaque subjective ones should be avoided. Complex weakness measure should be defined as repeatable multi-dimensional scores, based on more primitive objective measurements. When a weakness score is based on several dimensions, the individual dimensions should be made available as defined measures together with the final score, as they often provide useful features for further analytics.

The individual factors of the weakness score shall be discernable within the context of code analysis.
4.3. Incorporating architecture
There is more to mitigating risk than fixing reported weaknesses one by one in some order, guided by its rank and the underlying score. For example, the importance and the location of an asset must be taken into account. An asset – a system component - may be implemented by multiple modules, each with multiple weakness findings. Therefore, fixing a certain component can address multiple weaknesses within that component. This suggests that rank needs to be aggregated to components and assets.

4.4. Merging values from multiple sources
There is also a challenge in merging different rankings for the same entity. While many commercial and open source static code analysis tools are available today, no "uber tool" has emerged that would alone address the needs of the clients, nor is it likely that such a tool would emerge any time soon given the diversity of software systems, technologies, vulnerabilities and cyber attacks. Each tool in the market excels in certain types of findings and in order to provide assurance of their software, clients find themselves utilizing a combination of several tools [13].

Merging of reports from multiple sources also takes place in the NIST SCAP vulnerability management ecosystem. Why this is easier? Within the NIST SCAP ecosystem, multiple reports must agree on the identity of the vulnerability (using a nominal scale of universally agreed upon vulnerability identifiers, currently over 96,000 of individual ids have been assigned). A single value calculated using the CVSS score is associated with each vulnerability. CVSS is an adopted standard score for the disclosed vulnerabilities. Each vulnerability disclosure report may suggest the score, the final value is negotiated between all reports using a community process. Calculation of the CVSS score value is essentially a manual process, that involves vulnerability researchers. Then additional researchers review and correct the value until the final score is reached. In contrast, in the SCA approach a weakness is reported by a tool (which is the main promise of the SCA approach, that makes it economically attractive, systematic and objective). This means that the tool is responsible for calculating the score value.

While the upcoming standards like the OMG TOIF [17] and the OASIS SARIF [18] attempt to facilitate the agreement on weakness report identities reported by multiple SCA tools, normalizing the scores without relying on manual inputs is more challenging. Various SCA tools are using proprietary, not well-documented rankings, involving different scales, a mix of bounded and unbounded scores, scores with widely different factors. Merging the corresponding ranks or scores is therefore challenging. More specifically, the challenges include documenting scores, exchanging scores, merging scores from different sources. In the long term, a common score should emerge and standardization efforts should be focused at designing and promoting such score.

However, in the short and medium term, the options are limited to:
1. Providing a framework to take proprietary score values as input and normalizing them
2. Designing a common post-merge scoring system that does not rely on the score values provided by individual tools (for example, when there is none available), but uses other characteristics of weakness reports instead
4.5. Aggregating scores

Some scores allow better aggregation than others, and therefore it is also an open question whether to aggregate proprietary scores to assets first, or to merge into a common score before aggregating.

4.6. Avoiding Clustering

Bounded scores are prone to “high bias” (clustering of the high ranks) – the number of weaknesses with the top score tends to be still too high for any practical purposes leading to inability to act or making it too costly to proceed at all.

5. Weakness Measurements in OMG TOIF

This section describes the approach to a common exchange format for SCA tools that involves support for associating certain standard measures to weakness reports. The next section will describe an approach to a comprehensive definition of a weakness score, based on CVSS.

Tools Output Integration Framework (TOIF) defines the structured description of weakness findings and related elements. The objective of TOIF is to facilitate the normalization of weakness findings across multiple SCA tools and to define a common analytics environment which can be used to understand the security posture of a system under investigation by using multiple SCA tools. By design, TOIF facilitates development of standards-based, vendor neutral software as well as content for the common analytics environment. While focusing at normalizing the identities of weakness findings, TOIF provides support for normalizing weakness scores.

The key element of TOIF is called Finding. This element represents the raw weakness finding reported by one SCA tool. This element is the center of the web of related facts (all of which are provided separately from each other, the Finding element simply represents a unique entity that can be referenced by some facts).

TOIF directly addresses the situation where multiple SCA tools scan the same code base and focuses on the normalization of the weakness findings so that reports from multiple sources can be unified. As the result, TOIF allows prioritization of findings and reduction of false positive reports by focusing at the weaknesses reported independently by multiple tools.

To that end, TOIF includes a unique element called Weakness which represents the unified weakness, such that the weakness findings reported by multiple SCA tool can be treated as evidence linked to the weakness being claimed. Analytics tools built using the TOIF framework can perform the inferencing step that considers individual weakness reports and produces a unique Weakness element. This step involves the process known as evidential reasoning.

While Finding elements are produces by SCA tools and represent weakness finding report, the element Weakness represents an inferred weakness claim. Objects of this class are created by TOIF Analytics Tools, rather than by the SCA tools or the TOIF Adaptor Tools (which create the raw Finding objects). A Weakness object represents a unique weakness in the code of system.
Ranking Weakness Findings

under assessment, as supported by one or more Findings. This class supports integration and analysis of multiple TOIF reports and serves as the subject for additional statements, mainly the criticality, confidence, and citing statements.

TOIF provides several attributes for the raw weakness findings as well as for the inferred unique weakness element.

Finding element has two optional measurements:

- Confidence - specifies the confidence in this finding claim. Normalized bounded value 0-100 that uses a ratio scale.
- Criticality - specifies the importance of the finding in terms of the impact that it may cause. Normalized bounded score 0-100 that uses an interval scale.

Weakness element has two optional measurements:

- Confidence - specifies the inferred confidence in this weakness claim. This observation is calculated based on the individual Finding objects supporting this inferred Weakness object. Normalized bounded value 0-100 that uses a ratio scale.
- Criticality - specifies the inferred criticality of the weakness in terms of the impact that it may cause. Normalized bounded score 0-100 that uses an interval scale. This observation is calculated based on the individual Finding objects supporting this inferred Weakness object.

Weakness object may have one or more Citing statements associated with it. TOIF separated an inferred Weakness element with its metrics confidence and criticality from a Citing statement where a Citing statement is a further evaluation of the evidence related to a Weakness. Citing statements are usually produced by human security analysts. The corresponding Weakness element is merely a statement that one or more raw weakness findings are pointing to the same unique fault. A Citing statement is represented by the Citing element. Citing element has the following attributes:

- Confidence - specifies the confidence in this citing claim. Normalized bounded value 0-100 that uses a ratio scale.
- Verdict - specifies whether a Weakness is claimed to be a true or a false report.

Discussion

TOIF provides a mechanism for exchanging and aggregating two measures: confidence and criticality. TOIF does not provide an objective and repeatable approach to calculating these measures, but simply allow adding them to findings. This decision is aligned with the TOIF approach to focus at the lowest common denominator of SCA tools. It offers a simple but a long-term support for the community effort in providing structured definitions for confidence and criticality.

TOIF currently does not provide a mechanism for capturing diverse proprietary measures defined by the SCA tools, nor standardizes on any further common measures. While this is on the roadmap for TOIF, some of this work can be carried out in SARIF, in the context of the future alignment and interoperability between the TOIF and SARIF ecosystems.
6. Common Weakness Scoring System (CWSS)

This section describes a major scoring system for weakness findings, design of which is inspired by the CVSS, introduced earlier [10]. The next section will describe a companion specification that allows customization of the weakness score for a specific industry/domain by creating a “top-N-list” of weakness types for that domain, including some adjustments to the scores.

Common Weakness Scoring System (CWSS) is organized into three metric groups: Base Finding, Attack Surface, and Environmental. Each group contains multiple metrics - also known as factors - that are used to compute a CWSS score for a weakness.

- **Base Finding metric group**: captures the inherent risk of the weakness, confidence in the accuracy of the finding, and strength of controls.
- **Attack Surface metric group**: the barriers that an attacker must overcome in order to exploit the weakness.
- **Environmental metric group**: characteristics of the weakness that are specific to a particular environment or operational context.

6.1. Score Calculation

Each factor in the Base Finding metric group is assigned a value. These values are converted to associated weights, and a Base Finding subscore is calculated. The Base Finding subscore can range between 0 and 100. The same method is applied to the Attack Surface and Environmental metric group; their subscores can range between 0 and 1. Finally, the three subscores are multiplied together, which produces a CWSS score between 0 and 100.

6.2. Basic Finding Metric Group

The Base Finding metric group consists of the following factors:

- Technical Impact (TI)
- Acquired Privilege (AP)
- Acquired Privilege Layer (AL)
- Internal Control Effectiveness (IC)
- Finding Confidence (FC)

Technical Impact is the potential result that can be produced by the weakness, assuming that the weakness can be successfully reached and exploited. This is expressed in terms that are more fine-grained than confidentiality, integrity, and availability.

The Technical Impact should be evaluated relative to the Acquired Privilege (AP) and Acquired Privilege Layer (AL). The combination of values from Technical Impact, Acquired Privilege, and Acquired Privilege Layer gives the user some expressive power. For example, the user can characterize "High" Technical Impact with "Administrator" privilege at the "Application" layer.
If this set of values is not precise enough, CWSS users can use their own Quantified methods to derive a subscore. One such method involves using the Common Weakness Risk Analysis Framework (CWRAF) to define a vignette and a Technical Impact Scorecard. The Impact weight is calculated using vignette-specific Importance ratings for different technical impacts that could arise from exploitation of the weakness, such as modification of sensitive data, gaining privileges, resource consumption, etc.

The Acquired Privilege identifies the type of privileges that are obtained by an attacker who can successfully exploit the weakness.

Notice that the values are the same as those for Required Privilege, but the weights are different. In some cases, the value for Acquired Privileges may be the same as for Required Privileges, which implies either (1) "horizontal" privilege escalation (e.g. from one unprivileged user to another) or (2) privilege escalation within a sandbox, such as an FTP-only user who can escape to the shell.

The Acquired Privilege Layer identifies the operational layer to which the attacker gains privileges by successfully exploiting the weakness.

<table>
<thead>
<tr>
<th>Value</th>
<th>Code</th>
<th>Weight</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Critical</td>
<td>C</td>
<td>1.0</td>
<td>Complete control over the software being analyzed, to the point where operations cannot take place.</td>
</tr>
<tr>
<td>High</td>
<td>H</td>
<td>0.9</td>
<td>Significant control over the software being analyzed, or access to critical information can be obtained.</td>
</tr>
<tr>
<td>Medium</td>
<td>M</td>
<td>0.6</td>
<td>Moderate control over the software being analyzed, or access to moderately important information can be obtained.</td>
</tr>
<tr>
<td>Low</td>
<td>L</td>
<td>0.3</td>
<td>Minimal control over the software being analyzed, or only access to relatively unimportant information can be obtained.</td>
</tr>
<tr>
<td>None</td>
<td>N</td>
<td>0.0</td>
<td>There is no technical impact to the software being analyzed at all. In other words, this does not lead to a vulnerability.</td>
</tr>
<tr>
<td>Default</td>
<td>D</td>
<td>0.6</td>
<td>The Default weight is the median of the weights for Critical, High, Medium, Low, and None.</td>
</tr>
<tr>
<td>Unknown</td>
<td>UK</td>
<td>0.5</td>
<td>There is not enough information to provide a value for this factor. Further analysis may be necessary. In the future, a different value might be chosen, which could affect the score.</td>
</tr>
<tr>
<td>Not Applicable</td>
<td>NA</td>
<td>1.0</td>
<td>This factor is being intentionally ignored in the score calculation because it is not relevant to how the scorer prioritizes weaknesses. This factor might not be applicable in an environment with high assurance requirements; the user might want to investigate every weakness finding of interest, regardless of confidence.</td>
</tr>
<tr>
<td>Quantified</td>
<td>Q</td>
<td>This factor could be quantified with custom weights.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Code</th>
<th>Weight</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Administrator</td>
<td>A</td>
<td>1.0</td>
<td>The attacker gains access to an entity with administrator, root, SYSTEM, or equivalent privileges that imply full control over the software under analysis; or, the attacker can raise their own (lower) privileges to an administrator.</td>
</tr>
<tr>
<td>Partially-Privileged User</td>
<td>P</td>
<td>0.9</td>
<td>The attacker gains access to an entity with some special privileges, but not enough privileges that are equivalent to an administrator; or, the attacker can raise their own (lower) privileges to a partially-privileged user. For example, a user might have privileges to make backups, but not to modify the software's configuration or install updates.</td>
</tr>
<tr>
<td>Regular User</td>
<td>RU</td>
<td>0.7</td>
<td>The attacker gains access to an entity that is a regular user who has no special privileges; or, the attacker can raise their own (lower) privileges to that of a regular user.</td>
</tr>
<tr>
<td>Limited / Guest</td>
<td>L</td>
<td>0.6</td>
<td>The attacker gains access to an entity with limited or &quot;guest&quot; privileges that can significantly restrict allowable activities; or, the attacker can raise their own (lower) privileges to a guest. Note: this value does not refer to the &quot;guest operating system&quot; concept in virtualized hosts.</td>
</tr>
<tr>
<td>None</td>
<td>N</td>
<td>0.1</td>
<td>The attacker cannot gain access to any extra privileges beyond those that are already available to the attacker. (Note that this value can be useful in limited circumstances in which the attacker can escape a sandbox or other restrictive environment but still cannot gain extra privileges, or obtain access as other users.)</td>
</tr>
<tr>
<td>Default</td>
<td>D</td>
<td>0.7</td>
<td>Median of the weights for None, Guest, Regular User, Partially-Privileged User, and Administrator.</td>
</tr>
<tr>
<td>Unknown</td>
<td>UK</td>
<td>0.5</td>
<td>There is not enough information to provide a value for this factor. Further analysis may be necessary. In the future, a different value might be chosen, which could affect the score.</td>
</tr>
<tr>
<td>Not Applicable</td>
<td>NA</td>
<td>1.0</td>
<td>This factor is being intentionally ignored in the score calculation because it is not relevant to how the scorer prioritizes weaknesses. This factor might not be applicable in an environment with high assurance requirements that wants strict enforcement of privilege separation, even between already-privileged users.</td>
</tr>
<tr>
<td>Quantified</td>
<td>Q</td>
<td>This factor could be quantified with custom weights. Note that Quantified values are supported for completeness; however, since privileges and users are discrete entities, there might be limited circumstances in which a quantified model would be useful.</td>
<td></td>
</tr>
</tbody>
</table>
An Internal Control is a control, protection mechanism, or mitigation that has been explicitly built into the software (whether through architecture, design, or implementation). Internal Control Effectiveness measures the ability of the control to render the weakness unable to be exploited by an attacker. For example, an input validation routine that restricts input length to 15 characters might be moderately effective against XSS attacks by reducing the size of the XSS exploit that can be attempted.

When there are multiple internal controls, or multiple code paths that can reach the same weakness, then the following guidance applies:

- For each code path, analyze each internal control that exists along the code path, and choose the Value with the lowest weight (i.e., the strongest internal control along the code path). This is called the Code Path Value.
- Collect all Code Path Values.
- Select the Code Path Value that has the highest weight (i.e., is the weakest control).

This method evaluates each code path in terms of the code path's strongest control (since an attacker would have to bypass that control), then selects the weakest code path (i.e., the easiest route for the attacker to take).

According to CWSS, Confidence measure is related to the probability of a two-fold situation, that the reported issue:

1. is a weakness, and
2. can be triggered or utilized by an attacker
6.3. **Attack Surface Metric Group**

The Attack Surface metric group consists of the following factors:

- Required Privilege (RP)
- Required Privilege Layer (RL)
- Access Vector (AV)
- Authentication Strength (AS)
- Level of Interaction (IN)
- Deployment Scope (SC)

6.4. **Environmental Metric Group**

The Environmental metric group consists of the following factors:

- Business Impact (BI)
- Likelihood of Discovery (DI)
- Likelihood of Exploit (EX)
- External Control Effectiveness (EC)
- Prevalence (P)

6.5. **Final calculations**

A CWSS 1.0 score can range between 0 and 100. It is calculated as follows:

\[
\text{BaseFindingSubscore} \times \text{AttackSurfaceSubscore} \times \text{EnvironmentSubscore}
\]

The BaseFindingSubscore supports values between 0 and 100. Both the AttackSurfaceSubscore and EnvironmentSubscore support values between 0 and 1.

**Base Finding Subscore**

The Base Finding subscore (BaseFindingSubscore) is calculated as follows:

\[
\text{BaseFindingScore} = \left[ (10 \times \text{TechnicalImpact} + 5 \times (\text{AcquiredPrivilege} + \text{AcquiredPrivilegeLayer}) + 5 \times \text{FindingConfidence}) \times f(\text{TechnicalImpact}) \times \text{InternalControlEffectiveness} \right] \times 4.0
\]

\[f(\text{TechnicalImpact}) = 0 \text{ if } \text{TechnicalImpact} = 0; \text{ otherwise } f(\text{TechnicalImpact}) = 1.\]

The maximum potential BaseFindingSubscore is 100.

The definition of \(f(\text{TechnicalImpact})\) has an equivalent in CVSS. It is used to ensure that if the Technical Impact is 0, that the other added factors do not inadvertently generate a non-zero score.
TechnicalImpact and the AcquiredPrivilege/AcquiredPrivilegeLayer combination are given equal weight, each accounting for 40% of the BaseFindingSubscore. (Each generate a sub-value with a maximum of 10). There is some adjustment for Finding Confidence, which accounts for 20% of the Base (maximum of 5). The InternalControlEffectiveness can adjust the score downward, perhaps to 0, depending on the strength of any internal controls that have been applied to the issue. After application of InternalControlEffectiveness, the possible range of results is between 0 and 25, so the 4.0 coefficient is used to adjust the BaseFindingSubscore to a range between 0 and 100.

**Attack Surface Subscore**
The AttackSurfaceSubscore is calculated as follows:

\[
\text{AttackSurfaceSubscore} = \left[ 20\times(\text{RequiredPrivilege} + \text{RequiredPrivilegeLayer} + \text{AccessVector}) + 20\times\text{DeploymentScope} + 15\times\text{LevelOfInteraction} + 5\times\text{AuthenticationStrength} \right] / 100.0
\]

The combination of required privileges / access makes up 60% of the Attack Surface subscore; deployment scope, another 20%; interaction, 15%; and authentication, 5%. The authentication requirements are not given much focus, under the assumption that strong proof of identity will not significantly deter an attacker from attempting to exploit the vulnerability. This generates a range of values between 0 and 100, which are then divided by 100.

**Environmental Subscore**
The EnvironmentalSubscore is calculated as follows:

\[
\text{EnvironmentalScore} = \left[ (10\times\text{BusinessImpact} + 3\times\text{LikelihoodOfDiscovery} + 4\times\text{LikelihoodOfExploit} + 3\times\text{Prevalence}) \times f(\text{BusinessImpact}) \times \text{ExternalControlEffectiveness} \right] / 20.0
\]

\[
f(\text{BusinessImpact}) = 0 \text{ if BusinessImpact == 0; otherwise } f(\text{BusinessImpact}) = 1
\]

BusinessImpact accounts for 50% of the environmental score, and it can move the final score to 0. ExternalControlEffectiveness is always non-zero (to account for the risk that it can be inadvertently removed if the environment changes), but otherwise it can have major impact on the final score. The combination of LikelihoodOfDiscovery and LikelihoodOfExploit accounts for 35% of the score, and Prevalence at 15%.

**Discussion**

1. Many of the “dimensions” of CWSS are not discernable in the context of code analysis.
2. CWSS does not distinguish between different kinds of impact.


This section describes a specification that allows customization of the CWSS weakness score for a specific industry/domain by creating a “top-N-list” of weakness types for that domain, including some adjustments to the scores.
Common Weakness Risk Assessment Framework (CWRAF) provides a framework for scoring software weaknesses in a consistent, flexible, open manner, while accommodating context for the various business domains.

CWRAF benefits:
- Includes a mechanism for measuring risk of security errors ("weaknesses") in a way that is closely linked with the risk to an organization's business or mission.
- Supports the automatic selection and prioritization of relevant weaknesses, customized to the specific needs of the organization's business or mission.
- Can be used by organizations in conjunction with the Common Weakness Scoring System (CWSS) to identify the most important weaknesses for their business domains, in order to inform their acquisition and protection activities as one part of the larger process of achieving software assurance.

CWRAF and CWSS allow users to rank classes of weaknesses independent of any particular software package, in order to prioritize them relative to each other (e.g., "buffer overflows are higher priority than memory leaks"). This approach, sometimes referred to as a "Top-N list," is used by the CWE/SANS Top 25, OWASP Top Ten, and similar efforts. CWRAF and CWSS allow users to create their own custom Top-N lists.

By providing a repeatable way to customize the Common Weakness Scoring System (CWSS), CWRAF enables people to reason and communicate about the relative importance of different weaknesses. Users can automatically generate a more targeted specification of "Top-N" lists of weaknesses that are the most critical for the software that is used in the relevant business domains, missions, and technology groups. In conjunction with other activities, CWRAF ultimately helps developers and consumers to introduce more secure software into their operational environments.

The underlying concept that CWRAF and CWSS leverage is that in spite of there being over 600 weaknesses in CWE, all of them lead to one or more of only eight (8) technical impacts when they manifest in the operational system. In other words, if a particular CWE in a piece of software is exploited when that software is in use supporting a particular business it will result in the attacker being able to: modify data, read data, create a denial-of-service that results in unreliable execution, create a denial-of-service that results in resource consumption, execute unauthorized code or commands, gain privileges / assume identity, bypass protection mechanism, and/or hide activities.

By exploring how important these various technical impacts are to the business that the application is supporting we can flow these “importance” ratings to the CWEs that can result in those technical impacts.

The way CWSS scoring captures this business importance is by using the context of a vignette, which defines:
Ranking Weakness Findings

- A description of a system (or system-of-systems) that implement a business function using "technical archetypes" from various Technology Groups, such as web applications, industrial control systems, etc.
- A Business Value Context (BVC), which identifies the primary security concerns for deployed software that is covered by the vignette. The BVC describes potential harm that could occur to the business or mission if any weaknesses can be successfully exploited by an attacker, such as compliance failure, loss of reputation, or ecological disaster.
- A Technical Impact Scorecard, which lists the potential low-level effects of weakness exploitation (e.g., code execution or system crash) and ranks or prioritizes these impacts based on how they affect the performance of the business function being identified by the vignette.

Using the Business Value Context and the Technical Impact Scorecard, CWRAF provides vignette-specific input to CWSS, which can then be used to prioritize which weaknesses are of greatest concern and ideally must be addressed first.

Discussion

1. CWRAF is a useful approach to configure the ranking of weakness types for a given domain
2. CWRAF is a useful approach to link weakness types to risk for a given domain.
3. CWRAF does not address the issue of the underlying CWSS where a weakness type is not linked to a specific kind of impact, as in CVSS.

8. Weakness Reporting and Risk Management

This section describes the risk management context in which ranking are used. We explain how the rank generated by an SCA tool can contribute to the risk management, and demonstrate a gap between the original rank that can be established within the context of the code, and the ultimate rank in the risk management context.

Risk management is performed at the enterprise level within the business context. Risk to an enterprise is determined by the business impacts. Please refer to Figure 1 below for illustration. Some impacts are caused by the failures to either operational processes or to security processes, whether accidental or intentional. These failures may be caused by policies, processes, people, or technical systems. The architectural context is important for understanding processes, capabilities, and their dependencies that may lead to failures. An attack surface of the system is an architectural concept. Further, only some of the technical systems may be implemented in software. Since SCA tools analyze the code, they do not have the full context, including the architectural context and the business context.
If you are an information security practitioner, weakness ranking should guide you to those weaknesses that pose the greatest risk to your organization. Would you then select the worst issue – one with the highest rank - fix it and then move on the next one on the list? But then, given that 50,000 weaknesses have been reported, are you even making a dent in reducing the overall risk by addressing them sequentially? Are you actually addressing the problem of lowering the risk of your organization?

Finding and scoring weaknesses is a part of a larger need which is to estimate the risk to an organization. Individual weaknesses have an important role as components of risk.

How to measure weakness findings in such a way that this will help us to measure risk to the system?

SCA tools contribute to the understanding the security posture of the system under investigation. As illustrated in Figure 1, the flow of information goes from a weakness report by one or more SCA tools to a “confirmed weakness”, to a security vulnerability, which represents an exploitable weakness. Confirmed weaknesses may indicate vulnerabilities in the system, however from the risk assessment perspective, a vulnerability is manifested in a much larger context, which in addition to the code, includes policies, processes, people, hardware and additional software systems, which may not have been part of the original code assessment. A vulnerability is related to successful attacks at the entire deployed system. Penetration testing teams generate true vulnerability findings in the above sense. The benefit of the SCA approach is that under certain circumstances, confirmed weaknesses may be correlated with the vulnerabilities.
Assessing the security posture of the system involves understanding how the system can be attacked (or fail unintentionally), what characteristics of the systems make some attacks successful (which are by definition the vulnerabilities), what threat actors are capable and motivated to launch attacks, and what are the impacts of the attacks. Further, the security posture is the major contributor to the risk assessment of the enterprise that uses the system. The likelihoods of the attacks and the consequences of the attacks to the enterprise must be estimated. Based on the inventory of risks, mitigations are recommended.

Some of the mitigations involve fixing the vulnerabilities. Two things are important regarding mitigations. First, each mitigation has a certain cost to the enterprise. Second, mitigations are grouped into packages (often called mitigation options), by considering common architectural assets, cost and the combined risk reduction. The risk mitigation decision involves selecting an entire mitigation option. Thus, when translated back to the ranks of weakness finding reports, the rank of an individual finding is determined by the mitigation package, and much less by the score that such weakness may have individually. Moreover, some mitigations do not even involve code, making some code weaknesses irrelevant for the selected risk mitigation strategy despite their high rank from the code analysis perspective.

Figure 2 demonstrates that up to 8 different ranks is involved in the end-to-end risk management process.

In Figure 2 the original rank (1) may be generated by an SCA tool analyzing the code of a component implemented in software. Running multiple SCA tools produces different ranks (1,2). The capability to normalize and collate the findings by multiple SCA tools establishes a powerful analytics platform transcending the limitations of the code context in which the original SCA tools operate, by focusing at the “big data” analytic methods. When the original tool measurements are aggregated for each confirmed weakness, a normalized rank can be produced (3). This normalized rank takes advantage of the increased confidence when multiple independent tools identified the same weakness. Measurement related to the complexity of the finding, as well as the measurements based on the common facts related to the code base, can significantly increase the predictive power of the analytics environment.

Weaknesses provide evidence to risk assessments contributing to the vulnerability ranking (4). Technical impact characteristics and measurements of the exploitation complexity, if captured by the original SCA tools, can significantly improve the vulnerability ranking. However, the vulnerability ranking (4) is driven primarily by the architectural considerations, including analysis of the threat sources, their means and opportunities, attack surface, entry points, dependencies between system’s capabilities and processes, etc. In a larger context, not all system processes are implemented as code.
Figure 2 Multiple ranks in the context of risk management

Attack ranking (5) produced by the risk assessment captures these considerations. The risk ranking (6) is further based on the business considerations, such as the consequences of the successful attacks and their business impact. Risk rank (6) is based on the attack likelihood and severity of impact measurements.

Risk management uses the risk ranking to develop risk mitigations. The mitigation rank (7) is measured for a group of mitigations as a function of their combined reduction of risk, and cost. When traceability from risk and mitigation options to the original ranks of the weakness findings is preserved, the mitigation rank can be “projected” back to the weakness finding to determine their “ultimate” priority for fixing (8). Since the risk management decisions, captured in the mitigation rank (7) are done in a more comprehensive context than the initial weakness scanning, there can be a significant gap between the original rank and the “ultimate rank” driven by the risk management concerns.

On the other hand, Figure 2 demonstrates the contribution of the weakness measurements done by the original SCA tools to the risk management process. The important measurements from the viewpoint of the risk assessment and risk management, can be summarized as follows:

1. Capturing the category of technical impact of a weakness, if it is successfully exploited
2. Capturing the complexity measurement of a weakness finding
3. Capturing the magnitude of the exploit and the complexity of the exploit

Discussion
1. Weakness score is useful for weakness remediation, but is less useful for risk management, because the decision to fix a certain vulnerability is not local to the
characteristics of that vulnerability, but is determined by the selected mitigation option that involves the combined cost of fixing multiple related vulnerabilities, as well as the architectural importance of the assets.

2. Weakness remediation based on weakness score is not an effective way to mitigate risk

3. Weakness score should point to a class of attacks and likelihood of their success. But class of attacks is a taxonomy, not a score. It is often hidden as impact or criticality.

9. Measuring impact of a weakness

One of the essential characteristics of a weakness finding is the expected criticality of the issue indicated by this finding, which is related to the possibility of the exploitation of the issue and the magnitude of the impact. Several existing vulnerability/weakness scores define measures in this area.

CVSS uses a simplified Confidentiality/Integrity/Availability model to link vulnerabilities to impacts. Impact scores are calculated independently before they are aggregated into the overall score.

Confidentiality
The confidentiality (C) metric describes the impact on the confidentiality of data processed by the system.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>None (N)</td>
<td>There is no impact on the confidentiality of the system.</td>
<td>0.0</td>
</tr>
<tr>
<td>Partial (P)</td>
<td>There is considerable disclosure of information, but the scope of the loss is constrained such that not all of the data is available.</td>
<td>0.275</td>
</tr>
<tr>
<td>Complete (C)</td>
<td>There is total information disclosure, providing access to any / all data on the system. Alternatively, access to only some restricted information is obtained, but the disclosed information presents a direct, serious impact.</td>
<td>0.660</td>
</tr>
</tbody>
</table>

Integrity
The Integrity (I) metric describes the impact on the integrity of the exploited system.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>None (N)</td>
<td>There is no impact on the integrity of the system.</td>
<td>0.0</td>
</tr>
<tr>
<td>Partial (P)</td>
<td>Modification of some data or system files is possible, but the scope of the modification is limited.</td>
<td>0.275</td>
</tr>
<tr>
<td>Complete (C)</td>
<td>There is total loss of integrity; the attacker can modify any files or information on the target system.</td>
<td>0.660</td>
</tr>
</tbody>
</table>

Availability
The availability (A) metric describes the impact on the availability of the target system. Attacks that consume network bandwidth, processor cycles, memory or any other resources affect the availability of a system.
CVSS score suggests 3 separate dimensions for criticality, integrity and availability each with a simple grade measure describing the nature of the impact (e.g. full/partial/none). The grade measures are then mapped to numeric values for the combined score according to some weighting system.

In contrast, CWSS provides a single combined dimension called Technical Impact with a simple grade measure related directly to the magnitude of the impact (e.g. high/medium/low, etc.). This is described in a subsequent section in more detail.

Both approaches are subjective. Also, both approaches are suboptimal in the context of SCA, since code would in general provide a limited context in which the impact can be measured (by an automated capability). On the other hand, an important characteristic that is valuable for the analytics, is simply the kind of potential technical impact, i.e. a simple nominal measure with the values: Confidentiality impact (C), Integrity impact (I), Availability Impact (A), Confidentiality and Integrity Impact (CI), Confidentiality and Availability Impact (CA), Integrity and Availability Impact (IA), Full Impact (CIA).

The magnitude of the impact must be analyzed in a larger context (see below). Some of the complexity measures suggested earlier (magnitude and complexity of exploitation) also contribute to the final impact score.

10. Measuring confidence of a weakness

Some scores already take confidence in the weakness finding into account. For example, Fortify defines the Fortify Priority Order (FPO), a custom system of ranking the issues based on:

- Impact (a static value from 0 to 5)
- Probability (a static value from 0 to 5)
- Accuracy (a static value from 0 to 5)
- Confidence (a dynamic value from 0 to 5)

Likelihood is calculated dynamically based on Probability, Accuracy, and Confidence, and depending on the value of Likelihood and Impact, the issues are placed in either Critical, High, Medium, or Low buckets [12].

Using an opaque and subjective measure for confidence in an SCA tool and capturing the measurement in a weakness report is a good first step. However the benefit of such measure is very limited for subsequent analytics. An objective score for confidence should be used instead based on a collection of measurements for contributing factors.
Ranking Weakness Findings

CWSS score defines Confidence as a measure that represents the probability of the two-fold statement, namely that the reported issue:

3. is a weakness, and
4. can be triggered or utilized by an attacker

The measurement method is slightly more objective, and use the following definition of the measure:

<table>
<thead>
<tr>
<th>Value</th>
<th>Code</th>
<th>Weight</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proven True</td>
<td>T</td>
<td>1.0</td>
<td>The weakness is reachable by the attacker.</td>
</tr>
<tr>
<td>Proven Locally True</td>
<td>LT</td>
<td>0.8</td>
<td>The weakness occurs within an individual function or component whose design relies on safe invocation of that function, but attacker reachability to that function is unknown or not present. For example, a utility function might construct a database query without encoding its inputs, but if it is only called with constant strings, the finding is locally true.</td>
</tr>
<tr>
<td>Proven False</td>
<td>F</td>
<td>0.0</td>
<td>The finding is erroneous (i.e. the finding is a false positive and there is no weakness), and/or there is no possible attacker role.</td>
</tr>
<tr>
<td>Default</td>
<td>D</td>
<td>0.8</td>
<td>Median of the weights for Proven True, Proven Locally True, and Proven False.</td>
</tr>
<tr>
<td>Unknown</td>
<td>UK</td>
<td>0.5</td>
<td>There is not enough information to provide a value for this factor. Further analysis may be necessary. In the future, a different value might be chosen, which could affect the score.</td>
</tr>
<tr>
<td>Not Applicable</td>
<td>NA</td>
<td>1.0</td>
<td>This factor is being intentionally ignored in the score calculation because it is not relevant to how the scorer prioritizes weaknesses. This factor might not be applicable in an environment with high assurance requirements; the user might want to investigate every weakness finding of interest, regardless of confidence.</td>
</tr>
<tr>
<td>Quantified</td>
<td>Q</td>
<td></td>
<td>This factor could be quantified with custom weights. Some code analysis tools have precise measurements of the accuracy of specific detection patterns.</td>
</tr>
</tbody>
</table>

By definition, confidence is a report is related to the “trueness” of the finding. Why would an SCA tool report a situation that is not true? This is related to the complexity of automated control and data flow analysis of the code artifact performed by the SCA tools.

A tool would use a collection of indicators of weakness type, which lead the tool to specific “suspect” locations in the code. Each weakness type then has certain conditions that need to be satisfied for the report of the weakness to be true. These conditions usually involve conditions on the values of data from one or more data sources. Usually, an indicator is used to suggest the location of a potential weakness, and then control and data flow analysis is applied to collect the data values and investigate if the conditions are satisfied. Complexity of a finding is determined by the number of code paths that need to be examined in order to confirm conditions. Each code path may have a number of technical challenges that make control and data flow analysis difficult. This viewpoint on weakness composition is used in Software Fault Patterns (SFP) catalog, as well as in the well-known Juliette test suite for SCA tools [13,14]. According to this view, the technical challenges (or the code complexities) can be considered independently of the weakness types and their indicators. Note, that this is a logical model of the process, rather the description of the actual implementation in some SCA tool. In fact, SCA tools take innovative approaches to control and data flow analysis, with different coverage, accuracy and performance characteristics.

The separation of the indicator from the conditions, and the complexity of evaluating conditions on values during static analysis of code leads to the possibility of false reports, where a weakness is reported based on the indicator alone, or based on partially evaluated conditions. In many cases, depending on the nature of the indicator, this is a smart approach, because the indicator alone may be correlating with a serious problem that deserves attention.

Further, following the CWSS, a good weakness finding involves a separate analysis that it can actually be exploited by an attacker. This analysis involves:
Ranking Weakness Findings

1. knowledge of the target platforms, APIs that describe entry and exit points
2. control and data flow analysis techniques, similar to those used to confirm the satisfaction conditions at the location, identified by an indicator

Contributing factors to the confidence (trueness) of a weakness report are related to the following:

1. complexity of the weakness type,
2. complexity of the control and data flow for the location, implicated in the report
3. the nature of the technical challenges involved in the data solution
4. ability of the tool to overcome the challenges by applying a precise solution, or some kind of an approximate inference, and
5. complexity of control and data flow for the related to the opportunities for the exploitation of the issue found (which is usually a separate analysis).

11. Measuring complexity of a weakness finding

Complexity a weakness type can be measured as the number of data sources in a weakness. This can be measured using an ordinal scale with an interval of 0 to 7 or so. One of the most complex weakness types is a buffer overflow, where some variations may use up to 5-6 different data sources (identity of the buffer, length of the buffer, position in the buffer, length of the data being written, identity of the data, and identity of the function for performing the buffer access). This means that up to 6 different data sources must be tracked across various code paths to identify one path where the data values satisfy the “buffer overflow condition” – for example, that the length of the remaining space in the statically allocated buffer is less that the length of the data being written into the buffer.

Complexity of the control and data flow for the location of the potential weakness is determined by the number of code paths flowing into the identified location. One of the factors leading to the complexity of a finding can be objectively measured as the number of code paths using a ratio logarithmic scale.

Once the tool has identified one or more code path that constitute a solution for all data source (data values for the sources satisfy the weakness condition), the complexity of the entire solution must be calculated. Each of the data sources may have its own complexity.

An important measure of the complexity of one code path is the interprocedural length of the path, measured in the number of function calls between the identified value of the data source (e.g. a statement that assigns a constant to some variable) and the weakness location.

The measure of the complexity of findings should also include the nature of the technical challenges involved in the data flow computation. These technical challenges are described in a separate section below.

The key to objectively measuring confidence is to know which parts of the solution were achieved using precise methods, and which parts were approximated using some kind of
Ranking Weakness Findings

opportunistic inference to overcome the specific technical challenges. This can be measured as a ratio of the number of precisely calculated data values to the total number of data values involved in the data flow solution. When a solution does not involve data values, this measure will be defined as 1, indicating fully precise solution. A value of 0 can be interpreted as a fully approximated solution. A value of 0.5 means that one half of the data values were approximated, while the other half of the data values were computed precisely.

12. Measuring the exploitability of a weakness

The opportunities for the exploitation of the reported weakness require some additional analysis to connect the weakness to the known inputs of the code and to identify the possible data flow that exercises the control over the target code by injecting some data into the execution and by steering the computation in the desired direction. This can be objectively measured as follows:

1. the number of external inputs from which a weakness can be exploited
2. the magnitude of data exploitation (for each input path). The magnitude of data exploitation describes whether the attacker can freely inject data into the computation, or the input data is transformed so that only some parts of it are injected. This measure can use a simple grade scale with values: Full data injection, Filtered data injection, No data injection
3. the complexity of attacker’s control over the input path, which describes whether the attacker can simply inject the data, or conditions must be established by setting some portions of the data, or whether the conditions must be set prior to the injection using the same input stream, or using different input streams. This measure can use a simple grade scale with values: Direct injection, Injection with local conditions, Injection with prior local conditions, Injection with prior global conditions, No control

13. Measuring the challenges to the control and data flow solution

This section describes the technical challenges that a tool must overcome in order to provide a data flow solution in order to confirm some of the weakness reports. The number of code paths leading to a suspicious location gives one objective measure of the complexity of the finding. The other measure is related to the internal complexity of each code path that was analyzed in order to satisfy the conditions of the weakness finding in given location.

The following factors contribute to the complexity of a code path:

1. Interprocedural length of the code path (0 means that the code path is local to one function/method)
2. Number of conditions on the reported code path. In general, each condition requires an investigation of at least one additional data source, which can exponentially increase the amount of data flow analysis
3. Number of loops involved on the code path. While in general there is a difference between nested and consecutive loops on the code path from a data source to the
Ranking Weakness Findings

weakness location, the total number of loops may be an adequate measure of the complexity of the finding.
4. The specific code complexities that were encountered on the reported code path.

Code complexities can be described as follows:
1. Simple data propagation between assignments
2. Data propagation through expressions, including known library functions
3. Inferencing data values based on the conditions
4. Inferencing data values based on loops (resolving the values of variables that are changed in a loop)
5. Inferencing data values that are involved in (mutually) recursive functions
6. Resolving data structures, including pointers, arrays and structures
7. Resolving data from the objects
8. Resolving global variables
9. Resolving data originating from persistent resources
10. Resolving data coming from exceptions
11. Resolving data from multiple threads
12. Resolving data originating from inputs (including the input from the user/attacker, or from other executables)

According to the previous section, each code path can be resolved using precise methods or by performing an approximation. Approximations may result in false reports. Understanding the nature of the challenges will provide an essential feature for the analytics.

Complexity score of a code path is related to the well-known McCabe’s complexity score [15], however McCabe’s score is computed for the entire function, while the measure for a finding is measuring only the complexity of the code paths leading into the suspicious location. Consequently, it is possible to have a function with a high McCabe’s score, and a finding with a very low complexity score inside that function.

14. Requirements for weakness measurement system

This section outlines a set of requirements that together will lead to a good weakness measurement system. This white paper focuses on the role of standards and information exchange in this area.

In the long term, a common scoring system should emerge and standardization efforts should be focused at designing and promoting standard measurements for weakness findings that provide the foundations for the common scores. However, in the short and medium term, the options are limited to 1) providing a framework to take proprietary measurement results as input and normalize them, and 2) designing a common post-merge scoring system that does not rely on the measurement results provided by individual tools (for example, when there is none available), but uses other characteristics of weakness reports instead.
### Ranking Weakness Findings

<table>
<thead>
<tr>
<th>Ref</th>
<th>Group</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>G1</td>
<td>measurements</td>
<td>An SCA tool should output various measurements for a report item (weakness finding) according to own proprietary measure and capture these measurements in a standard format together with an identification of a measure itself. See also: G2, G4</td>
</tr>
<tr>
<td>G2</td>
<td>measures</td>
<td>An SCA tool should provide a standardized description of an implemented measure and make it available together with the weakness findings as metadata.</td>
</tr>
<tr>
<td>G3</td>
<td>measures</td>
<td>The SCA community should develop standardized descriptions of common measures and publish them as metadata. See also: G4</td>
</tr>
<tr>
<td>G4</td>
<td>measurements</td>
<td>An SCA tool may choose to implement a common measure a report item (weakness finding) and capture these measurements in a standard format together with an identification of the common measure. See also: G3, G5</td>
</tr>
</tbody>
</table>
| G5  | measures | Various common measures related to SCA weakness reports (including both standard and proprietary measures) shall be identified by unique and opaque identifiers. See also: G1, G4 and G2, G3  
*Note: The common measures include the ones available to the community either through de-facto implementation by the SCA tools, or through standard definitions.* |
| G6  | measures | Definitions of measures for SCA weakness reports shall be provided using a standard approach and one or more standard interoperable formats. 
*Note: interoperable formats should be based on industry standard notations such as XML, JSON, etc.* |
| G7  | measures | Definitions of measures shall be aligned with the OMG Structured Metrics Metamodel [16]. 
*Note: The OMG Structured Metrics Metamodel (SMM) defines a standard approach to building metrics libraries. SMM is described in the Appendix A to this document. SMM is adopted by industry, in particular, by the CISQ consortium. SMM defines an XML format, while a number of additional interoperable representations can be easily added.* |
| G8  | measurements | The common format for exchanging SCA results shall make standard and tool-specific measurements distinct in a more visible way than through the identification of the measure definition alone. 
*Note: For example, two distinct properties may be provided, one for standard measurements, and another one for tool-specific measurements.* |
<p>| M1  | metadata | Each measure definition shall include a version identifier. This provides an element of housekeeping (metadata) and allows management of multiple versions of the definition of the same measure in an enterprise environment |
| M2  | metadata | Each measure definition shall include the date when this definition was created/effected. This provides an element of housekeeping |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th>(metadata) and allows management of multiple versions of the definition of the same measure in an enterprise environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>M3</td>
<td>metadata</td>
<td>Each measure definition shall include the information about the author/contact person. This provides an element of housekeeping (metadata) and allows management of multiple versions of the definition of the same measure in an enterprise environment</td>
</tr>
<tr>
<td>M4</td>
<td>metadata</td>
<td>Each weakness finding measurements shall be linked to the corresponding definition of the measure using the unique identifier of the measure definition. This identifier shall refer to a specific version of the measure, and a mechanism shall be provided to link together multiple versions of the same (named) measure.</td>
</tr>
<tr>
<td>D1</td>
<td>definitions</td>
<td>Discernable objective measures should be preferred, and opaque subjective ones should be avoided. Complex weakness measure should be defined as repeatable multi-dimensional scores, based on more primitive objective measurements. When a weakness score is based on several dimensions, the individual dimensions should be made available as defined measures together with the final score, as they often provide useful features for further analytics.</td>
</tr>
<tr>
<td>D2</td>
<td>definitions</td>
<td>The individual factors of the weakness score shall be discernable within the context of code analysis.</td>
</tr>
<tr>
<td>C1</td>
<td>common measures</td>
<td>Impact of a weakness (simple enumeration without the measure of the magnitude of the impact)</td>
</tr>
<tr>
<td>C2</td>
<td>common measures</td>
<td>Number of data sources for a weakness type</td>
</tr>
<tr>
<td>C3</td>
<td>common measures</td>
<td>Number of code paths flowing into the identified location</td>
</tr>
<tr>
<td>C4</td>
<td>common measures</td>
<td>The interprocedural length of a code path from a data source to the weakness location</td>
</tr>
<tr>
<td>C5</td>
<td>common measures</td>
<td>The set of technical challenges that characterizes the code path</td>
</tr>
<tr>
<td>C6</td>
<td>common measures</td>
<td>For a weakness, the ratio between the number of data sources where the values were computed precisely to the total number of data sources (precisely analyzed as well as the ones analyzed using some approximation).</td>
</tr>
<tr>
<td>C7</td>
<td>common measures</td>
<td>The number of paths analyzed for exploitability (including 0, which means that such analysis was not attempted)</td>
</tr>
<tr>
<td>C8</td>
<td>common measures</td>
<td>The number of inputs from which the weakness can be exploited</td>
</tr>
<tr>
<td>C9</td>
<td>common measures</td>
<td>The magnitude of exploitability</td>
</tr>
<tr>
<td>C10</td>
<td>common measures</td>
<td>The complexity of controlling the computation for exploitation</td>
</tr>
<tr>
<td>S1</td>
<td>common score</td>
<td>Confidence in a finding as a score using c2-c10</td>
</tr>
<tr>
<td>S2</td>
<td>common score</td>
<td>Criticality of a finding as a score using c1,c7-c10</td>
</tr>
</tbody>
</table>
### Ranking Weakness Findings

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>S3</strong></td>
<td>common score</td>
<td>Complexity of a finding as score using c2-c6</td>
</tr>
<tr>
<td><strong>N1</strong></td>
<td>normalization</td>
<td>Weakness observation from multiple tools shall be integrated into a combined measure for the unique confirmed weakness. The structured definition for the merged measure shall be also made available.</td>
</tr>
<tr>
<td><strong>N2</strong></td>
<td>normalization</td>
<td>Weakness finding measures shall allow aggregation of measures across architectural components or assets. The structured definition of the corresponding collective measure shall be made available.</td>
</tr>
<tr>
<td><strong>N3</strong></td>
<td>normalization</td>
<td>Weakness finding measures shall support weakness prioritization.</td>
</tr>
<tr>
<td><strong>N4</strong></td>
<td>normalization</td>
<td>Weakness finding measures shall support risk analytics.</td>
</tr>
</tbody>
</table>
References

3. Merriam-Webster dictionary
   NIST SP-800-126 rev 3
12. SARIF communication, see also StackOverflow article “Using Fortify in project life cycle” https://stackoverflow.com/questions/34064429/using-fortify-tool-in-project-life-cycle#34097739
18. OASIS Static Analysis Results Interchange Format (SARIF), https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=sarif
Appendix A. Standard Exchange of Measures

Structured Metrics Meta-model (SMM) is a specification defined by the Object Management Group with the principle goal to provide a standard exchange of measures, measurements and observations [6]. This specification defines a meta-model for representing measurement related to structured model assets and their operational environments.

Given a structured model that represents the physical and logical assets at various levels of abstraction as entities and relations, SMM can be used to define the measures of these assets.

The SMM promotes a common interchange format that allows interoperability between existing tools, commercial services providers, and their respective models. This common interchange format applies equally well to development and maintenance tools, services, and models. SMM complements a common repository structure and so facilitates the exchange of data currently contained within individual tool models that represent modeled assets.

Initially, SMM was about measures for software artifacts but soon it was realized that the scope of the proposed approach is much more general, and is not limited to software. SMM was defined as a standard for the interchange of measure libraries and measurements over the any structured models, in particular ones defined according to the OMG MOF meta-model approach. Models of software are included into this definition, for example using another OMG specification – Knowledge Discovery Meta-model (KDM), which follows the OMG MOF approach to define structured models of software.

A measurement is not just a number and a unit of measure. The measured artifact must be indicated, the measure identified and contextual information retained as the observation. Contextual information is related by Observation, such as who, where, and when. Observation may serve to distinguish distinct utilizations of a given measure on a given measurand.

**SMM measure** - A method assigning comparable numerical or symbolic values to entities in order to characterize an attribute of the entities.

Measures are evaluation processes that assign comparable numeric or symbolic values to entities in order to characterize selected qualities or traits of the entities. Counting the lines of program code in a software application is one such evaluation.

There may be many measures that characterize a trait with differing dimensions, resolutions, accuracy, and so forth. Moreover, trait or characteristic may be generalize or specialized. For example, line length is a specialization of length that is a specialization of size.

Each measure has a scope, the set of entities to which it is applicable; a range, the set of possible measurement results; and the measurable property or trait that the measure characterizes. For example, the aforementioned line counting has software applications as one of its scope with line length as one of its measurable trait. Explicitly representing the scope and the measurable trait allows for the consideration of different measures, which characterize the same attribute for the same set of entities. Each measurable trait may have multiple, identifiably distinct measures.
The evaluation process may assign numeric values that can be ordered by magnitude relative to one another. These measures are modeled by the DimensionalMeasure class.

The evaluation process may alternatively assign numeric values that are percentages or, more generically, ratios of two base measurements. These measures are modeled by the RatioMeasure class. The percentage of comment lines in an application exemplifies this type of measure.

The evaluation process may also assign symbolic values demonstrating a grading that preserve the ordering of underlying base measures. These measures are modeled by the GradeMeasure class. Cyclomatic reliable/unreliable criterion illustrates one such grading. Reliable is comparably better than unreliable. Comparability is essential here because grading is not intended to model every possible assignment of measurands.

The documentations of measures, accomplished with measure libraries, should stand by themselves so that an interchange of measurements may simply reference such documentation and not duplicate it. The documentation of measures should also be precise and complete enough to provide for an unambiguous specification that can be executed on a referenced model.

Measures essentially fall into 2 “categories;” there are direct measures, which are measures that are taken directly against a measurand, and all others, which we shall call derived measures, as their result is based on some other measure(s), direct or derived.

The Measure class is central to the SMM specification. The Measure class models the specification of measures either by name, by representing derivations of base measures, or by representing method operations directly applied to the measured object. The essential requirement for the measure class is that it meaningfully identifies the measure applied to produce a given measurement. For example, McCabe’s cyclomatic complexity could be specified by its name, McCabe’s cyclomatic complexity, by a direct measurement operation or by rescaling counts of either independent paths or choice points. A measure may alternatively be identified by citing a library of measure which includes the measure by name.

SMM provides building blocks for structured definitions of measures. The elements are as follows:

- **NamedMeasure** – primitive dimensional measures which are well-known and can be specified simply by name. The meaning of applying the named measure should be generally accepted. For example, McCabe’s cyclomatic complexity.
- **RankingMeasure** – dimensional measure that represents (as does the GradeMeasure) simple range-based ranking or classifications based upon already defined dimensional measures. The result of each ranking is a value within a dimension and can be used as such. For example, one might use a RankingMeasure in mapping delivery time to “satisfaction” units. The delivery time satisfaction measurement can then be combined with other satisfaction measurements to get a customer total satisfaction measurement.
- **GradeMeasure** – a non-dimensional measure that represents simple range-based grading or classifications based upon already defined dimensional measures. Examples are:
  - Small, medium, large
CollectiveMeasure - measures that assess container entities by accumulating assessments of contained entities which are found by the base measure. Most engineering measures are collective. We count up lines of code for each program block and sum these values to measure routines, programs, and eventually applications. A similar process is followed to count operators, operands, operator and operand occurrences, independent paths, and branching points. Other frequently used container measures are based upon finding the maximum measurement of the container’s elements. Nesting depth in a program and class inheritance depth exemplify these collective measures.

BinaryMeasure - represents measures which when applied to a given entity accumulates measurements of two entities related to the given entity. For example, areas for two dimensional entities are often found by accumulating (multiplying) lengths.

DirectMeasure - DimensionalMeasure which applies a given operation to the measured entity.

CountingMeasure - DirectMeasure where the given operation returns 0 or 1 based upon recognizing the measured entity.

RatioMeasure - represents those measures that are ratios of two base measures. Examples include:
  - Average lines of code per module
  - Failures per day
  - Uptime percentage – Uptime divided by total time
  - Business data percentage of all data
  - Halstead level = Halstead volume divided by potential volume
  - Halstead effort = Halstead level divided by volume

A Ratio Measure and its two base measures frequently characterize three different traits of the same entity. If the dividend characterized the total code length of an application and the divisor characterized the number of program in the application, then the ratio characterizes the average code length per program. Ratios may also characterize traits of distinct entities. For example, a ratio may contrast the code length between a pair of programs.

RescaledMeasure - specifies a process that re-scales a measurement on an entity with one unit of measure to obtain a second measurement of the same entity with a different unit of measure. The RescaledMeasure class can also be used to apply a weight factor, to flip the sign, or make other linear adjustments. Non-linear adjustments may be specified with an Operation.

SMM allows definitions of complex definitions by combining the building blocks.

Figure 3 presents the fundamental approach of this specification. Measurement has a value conveying the measurement results.
**SMM measurement** - A numerical or symbolic value assigned to an entity by a measure. Measurement results are values from ordered sets. Such a set may be nominal (e.g., Poor, Fair, Good, Excellent) as long as there is an underlying order. A set may instead define a dimension where its values may be stated in orders of magnitude with respect to archetypal member. SMM allows for dimensional measurements. The magnitude is the measure’s unit of measure. SMM also allows for dimensionless measurements derived by ratios and grade schemes. In the former the ratio is derived from two measurements of the same dimension; whereas, in the latter measurements from a dimension are mapped to symbolic representations (e.g., 100-90 becomes “A”, 89-80 becomes “B”). The modeling of measurements mirrors the modeling of measure.

**SMM observation** – represents some of the contextual information which may be unique to this measurement such as date, measurer, and tool used. Measurements are sometimes repeated. An old carpentry rule is measure twice, cut once. To distinguish these multiple measurements, the observation and scope class can represent contextual information such as the time of the measurement and the identification of the measurement tool and the artifacts that are under measurement.

**SMM measurand** – An entity quantified by a measurement.

SMM avoids duplicating features of the measured artifact as features of the measurement. Consider as an example a log of bug reports. Possible measures are total bug count in the log, total time logged in the log, and bugs per time-period. The units of measures are a bug, a unit of time, and bugs per time interval, respectively. SMM does not provide representations for bug, start time and end time. Their representations must be provided elsewhere.

A measurement result is precisely identified only if its measure is identified. To understand the meaning of 1000 lines we need to know that it is the result of measuring a program’s length in

![Figure 3](image-url)
lines. The measured entity must be identified. That is, 1000 lines is for a particular program. Contextual information may also be needed. For example, function point counts of a program may vary depending upon the expert applying the measure.