Metrics for Requirements Management

J. Carlos Goti, Ph.D.
Rational Software Corporation
2800 San Tomas Expressway
Santa Clara, CA 95051-0951
(408) 496-8011
cgoti@rational.com

Date	Author	Description
Aug 10, 1998	Goti	First draft
Oct 10, 1998	Goti	Minor changes

Purpose

Record information about metrics applicable to Requirements Management as a result of a literature search. Record what sources were found and summarize the relevant contents.

Propose a starting list of RM measures or metrics.

Some ideas of the status of requirements metrics

A lot of the material reviewed does not address requirements metrics. Other requirements material (books and articles) mention that (obviously) we should measure stuff. They do not say what, specifically.

Examples:

- The INCOSE matrix that compares RM tools has no entry for metrics.
- [Gause 1989] addresses some generic measurements in the Technical Reviews chapter.
- [Weinberg 1997] also addresses generic measurements of process improvement, but he has no real substance on RM measures.

Some of what was found

[Paulk 1994] addresses the Capability Maturity Model and under “Requirements Management”, “Activity 3” – Changes to the allocated requirements are reviewed and incorporated into the software project, “Measurement 1” – Measurements are made and used to determine the status of the activities for managing the allocated requirements, says:

“Examples of measurements include:

- status of each of the allocated requirements:
- change and activity for the allocated requirements; and
- cumulative number of changes to the allocated requirements, including total number of changes proposed, open, approved, and incorporated into the system baseline.”

This is the only measurement under requirements management.

[US Army 1998]

“The STEP Metrics are described fully in Chapter 10 of the Department of the Army Pamphlet 73-7, February 1996. The Department of the Army has defined 12 metrics as applicable to all major AISs for which software is being developed, modified, or maintained. These metrics are divided into three categories:

- Management metrics
- Requirement metrics
- Quality metrics”

Under Requirements Metrics they show:

“These pertain to the specification, translation, and volatility of requirements. The two requirements metrics are:

- Requirements Traceability: This traces the linkage between the functional requirements to the software products including design and code and reports on the percentage of requirements traced.
- Requirements Stability: This tracks changes to the functional requirements and plots the cumulative number of change requests over time against the number of requests that have been resolved.”

[Roberts xxxx] reports work done by Dr. Noriaki Kano, member of the Japanese Union of Scientists and Engineers, the sponsors of the Deming Prize. The report is very short but shows some interesting ideas (not directly metrics though).

Requirements can be divided into three classes according to what users expect from the system being built:

- Expected: “Expected requirements are so obvious to the customer that they do not state requirements overtly. They are normally very obviously essential to the customer that stating these requirements is a bit silly. When these requirements are not met, the customer says nothing, and probably doesn’t even notice. When these features or services are not present, the customer complains. Continually improving on meeting these kinds of needs will not elicit customer loyalty or delight.

  Example: Telephone dialtone. If it is slow in coming or missing, customers are not happy. When it is present, the customer does not notice, much less become loyal to the provider.”

- Normal: “Sometimes referred to as “fundamental” quality. Customers overly state these needs and are quite cognizant of them. When these needs are met, customers are satisfied, when they are not met, customers are dissatisfied. For many types of requirements in this category, it is possible to deliver more than customer requires and generate additional perceived benefit.

Example: Price, performance, delivery."

- Delightful: "Customers have needs that they are not aware of. They are referred to as "latent" needs. They are real, but not yet in the customer's awareness. If these needs are not met by a provider, there is no customer response. They are not dissatisfied, because the need is unknown to them. If a provider understands such a need and fulfills it, the customer is rapidly delighted. Some articles describe this kind of need fulfillment as having "attractive" quality. It delights and excites customers and inspires loyalty.

  Example: A simple example is the 3M Post-it. The need for such a note posting tool has long been present, but not articulated until the product existed. It met a latent need, generated great enthusiasm, and became wildly successful."

This paper shows some crude measures (there may be more to this method than the short paper describes) to attempt to categorize requirements as explained above. This would allow product feature planning maximizing customer satisfaction.

[Pfleeger 1998] is a very complete and modern book on Software Engineering with lots of academic and practice references. It has a solid chapter called "Capturing the Requirements" and some recommendations about measuring requirements.

The author proposes measures about the requirements themselves and about the RM process.

The first category involves "requirements size"; that is, the number of requirements as they vary in time. We can expect the number of requirements to grow over time and stabilize as we approach delivery of the product. The number of requirements should be measured by requirement type.

Similarly, we can measure changes to requirements over time. This measure would give us an idea of the "stability" of our whole set of requirements.

Since requirements are to be used by designers and testers (I would add domain expert and users to this list and have a set of questions for them also), we can get a subjective measure of their goodness by having them "rate" each requirement in one of the following categories. We can keep track of how many fall in what category and assess the "goodness" of our set of requirements.

Categories for designers:

- "1 – means that you understand this requirement completely, you have designed from similar requirements in the past, and you should have no trouble developing a design from this requirement."
- "2 – means that there are elements of this requirement that are new to you, but they are not radically different from requirements you have successfully designed from in the past."
- "3 – means that there are elements in this requirements that are very different from requirements you have designed from in the past, but you understand it and think you can develop a good design from it."
- "4 – means that there are parts of this requirement that you do not understand, and you are not sure that you can develop a good design."
- "5 – means that you do not understand this requirement at all, and you cannot develop a design from it."

Categories for testers:

- "1 – means that you understand this requirement completely, you have tested against similar requirements in the past, and you should have no trouble testing the code against this requirement."
- "2 – means that there are elements of this requirement that are new to you, but they are not radically different from requirements you have successfully tested against in the past."
- "3 – means that there are elements of this requirements that are very different from requirements you have tested against in the past, but you understand it and think you can test against it."
- "4 – means that there are parts of this requirement that you do not understand, and that you are not sure that you can devise a test to address this requirement."
- "5 – means that you do not understand this requirement at all, and you cannot develop a test to address it."

Initial proposal

Based on the literature search, prior knowledge on measurements and metrics we can propose the following initial (and simple) ideas.

Measurements have practical value when compared to prior measurements (to determine trends) for either the same project or different projects.

At the inception phase, we should develop estimates for requirements measurements (and hence estimates for subsequent artifacts (design, code, test, documentation, etc.) to assess the cost of the project.

As requirements evolve:

- We should measure the current set of requirements for size (can we call this requirements complexity?)
- We should measure RM process values:
  - For the elicitation phase we can measure requirements size (this will tell us how much progress we are making).
  - When requirements are approved by users (baselined) we can measure how much they change over time (to establish requirements churn)
- We should measure how requirements trace to design elements
- We should measure how requirements trace to test elements
- We should measure the "goodness" of requirements for users
- We should measure the "goodness" of requirements for design
- We should measure the "goodness" of requirements for test

Let's assume that "iterations" are the time periods of interest for requirements measurements. That is, we will track requirements metrics from iteration to iteration to establish progress.

Let's assume the following ReqPro (project) structure:

```
Features
        Use-cases
                Events-in-the-flow
                        Test-requirements
                Scenarios (design-elements)
```

Features are defined as in the RMUC. Features support user needs. A system has many features.

Use-cases are really handles to a use case and point to the use-case name (and brief description). Use-cases are in support of features. There are many Use-cases per feature.

Events-in-the-flow are the basic or alternate steps in the flow of a use case. They relate to a given Use-case. There are multiple Events-in-the-flow per Use-case.

Test-requirements are descriptions of how a given Event-in-the-flow is going to be tested. They relate uniquely to an Event-in-the-flow. Probably, there is one Test-requirement for each Event-in-the-flow.

Scenarios are design-elements (as in ROSE) that elaborate a given Use-case. They relate uniquely to a given Use-case. There is more than one Scenario per Use-case.

Size metrics

Project size: Number of Features + Number of Use-cases (modified by the number of Events-in-the-flow and the number of Scenarios) – weights needed to develop a single figure.

Another possibility is to estimate Project size using a multiple set of figures (www Features, xxx Use-cases, yyy Events-in-the-flow, and zzz Scenarios).

Number of Features

Number of Use-cases

For estimates: average number of Events-in-the-flow for all Use-cases

For actuals: real number of Events-in-the-flow for a given Use-case

For estimates: average number of Scenarios per Use-case

For actuals: real number of Scenarios per Use-case

Initial estimates:

At the end of inception there is a pretty solid list of Features. Instead of using high, medium, low for the complexity attribute of a feature we can use the estimated number of Use cases and develop an average number of Steps-in-the-flow and Scenarios. This will give us an initial estimate.

Measuring progress:

As the project progresses, we compare estimates and averages with actuals and start tracking progress from iteration to iteration.

When a Use-case is identified, estimates of its number of Events-in-the-flow and Scenarios are developed. They should get us closer to the actual metrics until they are fully defined.

Goodness metrics

It does not seem useful to estimate measures of goodness before the artifact is defined.

For textual information, the research shows that useful measures are:

- Word count - textual length: this measure is conceptually similar to lines-of-code for programs and should indicate the complexity of the element.
- Fog index: this measure of grammatical complexity similar to cyclomatic complexity for programs.
- Target user understanding: this is an objective measure of clarity and potential usefulness. See the categories of understanding (1 to 5) listed above in the [Pleeeger 1998] reference.

In practice, it seems that fog index and target user understanding are the most appropriate to our situation.

For Features: user/domain expert understanding rating and fog index.

For Use-cases: designer understanding rating and fog index for the entire document (all sections)

For Events-in-the-flow: tester understanding

Traceability metrics

As soon as an element is defined we can estimate "how many" elements it should connect to in a downstream fashion. This is equivalent to establishing an estimate of its complexity.

For example:

- Given that we have identified a Feature, we should be able to estimate how many Use-cases it will need.
- Given that we have identified a Use-case, we can estimate how many Events-in-the-flow it will need and how many Scenarios it will need.

Hence, in each iteration assessment phase we can adjust our estimates and compare our estimates to actuals in a progressive manner. The estimates tell us how much work we have left and as items get completed we can report actuals VS estimates.

Traces from Use-cases to Events-in-the-flow can be obtained from the RM tool.
Traces from Use-cases to Scenarios may need to be imported from Rose for actual figures.
Traces from Events-in-the-flow to Test-requirements can be obtained from QA personnel as they get completed.

Change metrics

It seems that there are two types of changes we may want to track:

- Changes in content: some requirement needed to be modified for reasons having to do with user needs or reasons having to do with project difficulties. Both of these changes have the effect of forcing the project to evaluate impacts and possibly re-direct some efforts.
- Changes due to progress: some things are getting done and we need to now about them. For example a Use-case got approved, or completed, or…

The project would want to track both types. The details and reporting schemes need to be matched to the process, but we may want to know, at the end of an iteration:

How many Features were modified.

How many Test-requirements passed.

How many Use-cases were exercised and approved by the user (not on paper but in real execution mode).

Etc.

Some references from the literature search

Most books on software metrics either ignore of have almost nothing on requirements metrics.

[Gause 1989] – Gause, Donald C. and Weinberg, Gerald M., “Exploring Requirements: Quality Before Design”, Dorset House Publishing, 1989.

[Halligan 1993] - Halligan, R., "Requirements Metrics: The Basis of Informed Requirements Engineering Management," *1993 Complex Systems Engineering and Assessment Technology Workshop,* Naval Surface Warfare Center, Dahlgren, Virginia, July 1993. This was found in the Alan Davis bibliography, but I was not able to find the paper itself.

[Paulk 1994] – Paulk, Mark C., Weber, Charles V., Curtiss, Bill, and Chrissis, Mary Beth, “The Capability Maturity Model: Guidelines for Improving the Software Process”, Addison-Wesley, 1994.

[Peng 1993] – Peng, Wendy W. and Wallace, Dolores R., “Software Error Analysis”, NIST Special Publication 500-209, March 1993. Found in Internet URL?

[Pfleeger 1998] – Pfleeger, Shari Lawrence, “Software Engineering – Theory and Practice”, Prentice Hall, 1998.

[Roberts xxxx] – Roberts Information Services, “Understanding Customer Requirements: Development of Metrics (Kano Model)”, Internet URL http://www.servqual.com/RIS/7.1.html.

[US Army 1998] – US Army Operational Test and Evaluation Command, “Software Test and Evaluation Panel (STEP) Metrics”, last updated January 27, 1998. Internet URL http://www.usace.army.mil/inet/functions/im/lcmis/lcmstep/lcmstep.htm.

[Weinberg 1997] – Weinberg, Gerald, M., “Quality Software Management – Volume 4 – Anticipating Change”, Dorset House Publishing, 1997.