Quantifying Quality Requirements Using Planguage

Erik Simmons
Intel Corporation
JF1-46
2111 NE 25th Ave.
Hillsboro, OR 97214-5961
erik.simmons@intel.com

Version 1.1, 03/30/01

Prepared for Quality Week 2001

Key Words: Product Quality; Quality Requirements; Quantified Quality

Author Biography

Erik Simmons has 15 years experience in multiple aspects of software and quality engineering. Erik currently works as Platform Quality Engineer in the Platform Quality Methods group, part of the Corporate Quality Network at Intel Corporation. He is responsible for Requirements Engineering practices at Intel, and lends support to several other corporate software and product quality initiatives. Erik is a member of the Pacific Northwest Software Quality Conference Board of Directors. He holds a Masters degree in mathematical modeling and a Bachelors degree in applied mathematics from Humboldt State University in California.

Abstract

Within the last decade, requirements engineering has benefited from increased attention. Several good books are now available, from general textbooks on requirements engineering to specific monographs on advanced topics. Among the many benefits has been an increased awareness of the importance of specifying quality requirements. However, outside of structured English, few methods for specifying quality requirements have been established. Planguage, created by Tom Gilb, is one notable exception. Designed to quantify qualitative statements in plans, specifications, and designs, Planguage is a keyword-driven language that allows measurable, testable quality requirements to be written. Planguage has many benefits; it is easy to learn, flexible, compact, extensible, and prevents omissions by providing a consistent set of parameters for quality requirements. In this paper, Planguage keywords and syntax are introduced. Examples of quality requirements before and after using Planguage are given, and the experiences of introducing Planguage within a product engineering environment are discussed.

Introduction

The last decade has seen an increased focus on the methods, process, and benefits of good requirements engineering. In the past few years alone, several very good books have been published on the topic. Undergraduate and graduate programs now more commonly introduce students to the fundamental concepts and techniques of requirements engineering.

Despite these and other advances, few techniques are taught for properly specifying quality attributes like performance, reliability, scalability, and ease of use. In most cases, structured English sentences are used to express the underlying requirements using terms that are difficult or impossible to test adequately. Qualitative terms like *easy, fast, reliable, secure, scalable, efficient, robust,* and a host of others are fertile ground for misunderstandings between product stakeholders.

Planguage was created by Tom Gilb in order to overcome these problems by quantifying qualitative terms [Gilb01, Gilb97a, Gilb97b]. Planguage is a keyword-driven language whose name is derived from a contraction of the words planning and language[1]. Planguage can be used in requirements specifications, design documents, plans, and other places where qualitative statements are common. Its primary benefits are quantifying the qualitative and improving communication about complex ideas. In addition to these, Planguage has several other desirable features and benefits:

Ease of Learning and Use

Planguage can be taught effectively to individuals and groups in a short period. At Intel, Planguage is covered in only a few hours as part of the requirements engineering curriculum. Although this brief exposure is not enough to guarantee successful adoption and use of Planguage, when combined with a small amount of follow-up mentoring and a catalog of examples the results have been quite good. More than 1,200 students at Intel have been exposed to Planguage within the past 12 months, and Planguage has made its way into many product development efforts. It is used by engineering, quality assurance, marketing, and program management alike in a widening array of documents, plans, and designs.

Flexibility and Extensibility

Planguage is designed to be extensible and customizable to fit local needs. This includes the addition of keywords and the rich structure of Planguage, with its ability to create and label statements, collections, and other internal structures for reuse. These properties have made Planguage popular and useful across differing product development efforts – an essential capability in order to obtain broad adoption and use in as diverse an environment as Intel.

Prevention of Omissions

One of the most powerful benefits of Planguage is its ability to prevent omissions when quantifying qualitative statements. Because keywords are prescribed for all the important dimensions, users of Planguage are less likely to omit necessary information. Planguage is equally effective in this regard whether implemented as a table within a document or as part of an automated requirements repository. In both cases, users praise its ability to bring issues to light through its complete, separate, and consistent treatment of the important dimensions of quantification.

Separation of Success and Survival

When considering qualitative concepts, there are usually many levels of achievement (or a range of achievement) possible. The question is not whether a system is reliable or secure, but *how* reliable or secure. Planguage excels at expressing these ideas through its use of more than one level of achievement. By allowing for specification of the best recorded level of performance, the

[1] The term Planguage is also used as the name of some programming languages for parallel processors, but that use is not related to its use in this paper.

optimum level, the planned level, and the level below which financial or political failure occurs, Planguage paints a detailed and complete picture of success and survival, allowing for informed, due-diligent decision making.

Planguage Keywords & Syntax

Planguage has a rich set of keywords. The commonly used keywords are given in Table 1.

Table 1: Planguage Keywords

TAG	A unique, persistent identifier
GIST	A short, simple description of the concept contained in the Planguage statement
STAKEHOLDER	A party materially affected by the requirement
SCALE	The scale of measure used to quantify the statement
METER	The process or device used to establish location on a SCALE
MUST	The minimum level required to avoid failure
PLAN	The level at which good success can be claimed
STRETCH	A stretch goal if everything goes perfectly
WISH	A desirable level of achievement that may not be attainable through available means
PAST	An expression of previous results for comparison
TREND	An historical range or extrapolation of data
RECORD	The best-known achievement
DEFINED	The official definition of a term
AUTHORITY	The person, group, or level of authorization

As an example of the extensibility of Planguage, four sub-keywords have been created for the keyword METER. The sub-keywords are designed to add precision and specificity to the METER statement, and are given in Table 2.

Table 2: Sub-keywords for the METER Keyword

METHOD	The method for measuring to determine a point on the Scale
FREQUENCY	The frequency at which measurements will be taken
SOURCE	The people or department responsible for making the measurement
REPORT	Where and when the measurement is to be reported

Besides keywords, Planguage also offers several convenient and useful sets of symbols:

- Fuzzy concepts requiring more details are marked using angle brackets: <fuzzy concept>
- Qualifiers, which are used to modify other keywords, are contained within square brackets: [when, which, …]
- A collection of objects is indicated by placing the items in braces: {item1, item2, …}
- The source for a statement is indicated by an arrow: Statement ← source

Using Qualifiers

Qualifiers allow for precise description of conditions and events. They add richness, precision, and utility to Planguage. Here are several (unrelated) examples of qualifier use:

PLAN **[Q1 ’00]**: 20,000 units sold
MUST **[First year]**: 120,000 units sold

WISH **[First release, enterprise version]**: 1 Dec. 2000
PLAN **[US market, first 6 months of production]**: Defects Per Million < 1,000

METER **[Prototype]**: Survey of focus group

METER **[Release Candidate]**: Usability lab data

A Basic Application of Planguage

Requirements often contain statements like the following:

"The system must be easy to learn."

When presented with this first requirement, nearly everyone would agree that it is not testable as written. It is up to the tester or someone else downstream to decide what "easy" is, what "learn" means, and how to test whether the product meets minimum levels of goodness.

A second common form of the statement of usability is made in structured English:

"The system must be used successfully to place an order in under 10 minutes without assistance by at least 80% of test subjects with no previous system experience."

This is an improvement over the first requirement, and represents the typical state of the practice. The second wording gets a better response for testability, and many believe that they could write and execute tests for it.

Here is the Planguage version:

TAG: Learnable
GIST: The ease of learning to use the system.
SCALE: Time required for a Novice to complete a 1-item order using only the online help system for assistance.
METER: Measurements obtained on 100 Novices during user interface testing.
MUST: No more than 7 minutes 80% of the time
PLAN: No more than 5 minutes 80% of the time
WISH: No more than 3 minutes 100% of the time
PAST [our old system]: 11 minutes ← *recent site statistics*
Novice: DEFINED: A person with less than 6 months experience with Web applications and no prior exposure to our Website.

This statement provides a great deal of information in a compact format. Additionally, it is testable and far less ambiguous than the previous structured English statement.

Finding Scales and Meters

Scales exists for just about any concept. Here are some helpful hints for locating/defining scales:

- Divide the measured quality into its elementary components first if possible
- Use known, accepted scales of measure when possible
- Derive new scales from known scales by substituting terms
- Incorporate qualifiers in the scales to increase usefulness and specificity
- Don't confuse scale with meter
- Share effective scales with others

Examples of scales for several situations are given in Table 3:

Table 3: SCALE Examples

Environmental Noise	dBA at 1.0 meter
Software Security	Time required to break into the system
Software Maintainability	Average engineering time from report to closure of defects reported prior to release

System Reliability #1	The Mean Time To Failure of the system
System Reliability #2	The time at which a certain percentage of the system failures have occurred (known as the B-life). For example, at the B10 life, 10% of the units have failed.
System Learnability	Average time for <novices> to become <proficient> at a defined set of tasks (this can be measured on competing prototypes)
Vendor Of Choice	Gaps between customer's expressed importance and satisfaction for various product and service attributes
Revenue	Total sales in US$, Average Selling Price, etc.
Market Share	Percentage of Total Available Market (TAM)

To locate a meter, study the scale carefully. If no meter comes to mind:

- Look at references, handbooks, examples, etc. for ideas
- Ask others for their experience with similar methods
- Look for examples within test procedures

Once you have located a candidate meter, be sure that:

- The meter is adequate in the eyes of all stakeholders
- There is no less-costly meter available that can do the same job (or better)
- The meter can be measured *before* product release or completion of the deliverable

Examples of Meters for several situations are given in Table 4:

Table 4: METER Examples

Environmental Noise	Lab measurements performed according to the Environmental Test Handbook
Software Security	An attempt by a team of experts to break into the system using commonly available tools
Software Maintainability	Analysis of at least 30 consecutive defects reported and corrected during development
System Reliability #1	A Probability Ratio Sequential Test demonstration with α=10%, β=10%, Discrimination Ratio = 3
System Reliability #2	Weibull analysis of 50 sample units bench tested to failure

Planguage Examples

In practice, the TAG keyword is often dropped, as is the GIST keyword. Instead, the tag itself is placed before the text of the gist, like this:

LEARNABLE: The ease of learning to use the system

instead of

TAG: Learnable
GIST: The ease of learning to use the system

Most of the examples that follow use the shorter format combining the tag and gist.

Example 1: Power Consumption
Before Planguage, here is an actual requirement as written. Only the company names have been altered:

“The third key requirement is power consumption. Generally, the power consumption requirements are driven by noise requirements, or CE compatibility. The customers expressed the need for lower active power consumption so that passive cooling can be used. However, this is one possible implementation, and other implementations need to be addressed by engineering. Standby power consumption should meet the levels obtained by CE devices; 5-10W, and be achievable with the fan off. Cost is a factor. 10W standby is acceptable if the implementation cost is less than that of 5W standby. These requirements were articulated by *Company1*, *Company2*, *Company3*, *Company4*, and *Company5*.”

The same requirement written using Planguage:

STANDBY: Standby Power Consumption ←{Company1, Company2, Company3, Company4, Company5}
GIST: The amount of power consumed by the system with the fan off and the HDD not spinning
SCALE: Watts
METER: Measurement on 3 units for 10 seconds at 23°C, ± 2°C
MUST: 10W
PLAN[CostOK]: 5W
CostOK: Design and manufacturing costs do not exceed 10W cost by more than 25%
NOTE: Relates to noise and CE compatibility requirements. Passive cooling within the system is desired.

This rewritten statement is traceable (since it is uniquely and persistently identified by its TAG), measurable (and testable), and more precise than the original while taking up less space and using fewer words than before.

Example 2: Acoustic Noise

Another actual requirement, as originally written:

“The second key requirement is that the acoustic noise generated by the PC be at levels similar to common consumer electronics equipment. Based on OEM feedback, this acoustic noise level while the PC is active (HDD active) needs to be in the range of 25-33dB. *Company1* shared the progress they have made in this area. They have moved from 38dB active in 1996 to 33dB active in 1997. Their goal is to maintain less than 33dB. *Company2’s* requirement is 25dB during active state.”
Rewritten using Planguage:

NOISE: Acoustic Noise ←{Company1, Company2}
GIST: The amount of acoustic noise generated by the system with the fans running and HDD spinning.
SCALE: dBA
METER: Acoustic Sound Pressure test from the current Environmental Test Handbook, measured on 3 units
MUST [Company1]: 33dBA
MUST [Company2]: 25dBA
PLAN: 25dBA
TREND [1996 – 1997, Company2]: 38dBA → 33dBA

Note that other solutions are possible. The original requirement does not make clear whether the PLAN should be 33dB, 25dB, or some other value. Similarly, the MUST statement(s) could be written in several other ways. It is the conversations required to determine which expression is correct that are valuable.

Example 3: Development Process Efficiency

This example makes use of the optional sub-keywords for the Meter (Method, Frequency, Source, and Report).

EFFICIENT: The efficiency of the development process
SCALE: Rework as a percentage of total effort expended
METER: Examination of defect logs and project data
METHOD: Total Rework (defect logs) divided by total effort (project tracking database)
FREQUENCY: Measured monthly
SOURCE: Software Process Engineering Team data
REPORT: Senior Staff Meeting
MUST: No more than 45%
PLAN: No more than 35%
PAST: 50-60% ← guess, based on industry averages.

Example 4: Software Scalability

Scalability.CPU: The CPU usage pattern under increasing application stress.
SCALE: Minimum application transactions per second required to sustain 100% CPU utilization for at least 15 seconds.
METER: Stress testing of the application using automated software drivers and a representative operational profile.
MUST [Single Processor, 500MHz]: At least 45 TPS
PLAN [Single Processor, 500 MHz]: At least 60 TPS

Example 5: Security

This example illustrates how fuzzy concepts can be marked as needing clearer definition. The requirement could be used as a template for several projects, with the terms and achievement levels defined as needed for each one:

Security.Access: The resistance of the system to <unauthorized access>.
SCALE: Time required to obtain <unauthorized access> to the system using commonly available tools and techniques.
METER: Attempted <access> by a team of two skilled security engineers with no special knowledge of the system.
PLAN: At least 16 hours
MUST: At least 8 hours

Example **6: Memory Use**

TAG: MemoryUse
GIST: The amount of memory used by the application.
SCALE: Megabytes
METER: Performance Log observations made during system testing.
PLAN [Peak committed memory, Representative Operational Profile]: No more than 24 MB
PLAN [Peak committed memory, Stress Profile]: No more than 40 MB
PLAN [Average committed memory, Representative Operational Profile]: No more than 16 MB
PLAN [Average committed memory, Stress Profile]: No more than 24 MB
Representative Operational Profile: DEFINED: An operational profile that is likely to occur during use of the system after deployment. Specifically not a profile designed to stress the application in ways not possible or rarely encountered in actual use.
Stress Profile: DEFINED: An operational profile designed to cause extreme resource consumption or challenge the system's performance, regardless of whether the profile is likely or even possible to occur in actual use.

Lessons Learned Introducing Planguage at Intel

Planguage has been among the most popular topics in the requirements engineering coursework taught at Intel. The material has been presented to a broad cross section of the company, in terms of both job function and geographic location. Students embrace Planguage because it solves a real problem with elegance and simplicity. Most teams have felt the pain of mismatched expectations that stemmed from weak, qualitative terms. Planguage presents an opportunity to avoid those problems from the start. Test teams and quality assurance personnel also like the clarity and accountability that comes with Planguage requirements.

If students have any difficulty as they learn Planguage, it is usually when they first attempt to locate scales and meters for Planguage statements. Students sometimes confuse scale and meter, so a simple example such as natural gas service or residential water supply is useful and provides a way to clarify thinking for less-obvious situations.

Although Planguage is a simple concept that has innate appeal, students typically require some additional assistance before they become independently proficient with the techniques involved (especially scales and meters). Two strategies work well to provide this assistance: follow-on mentoring from experienced Planguage users and a catalog of example Planguage requirements from which to draw ideas and templates. This catalog can be extended with new material as it is developed, and could be nicely implemented as a Website.

Planguage is designed for a much broader application than just quality requirements. Once Planguage use has been established on a team or in a business unit, others pick the language up for roadmaps, marketing objectives, vision statements, plans, and other uses. The positive benefits of such cross-pollination are significant.

References

Gilb01	Gilb, Tom, *A Handbook for Systems & Software Engineering Management using Planguage*, Addison Wesley 2001
Gilb97a	Gilb, Tom, *Requirements-Driven Management: A Planning Language*, Crosstalk, June 1997
Gilb97b	Gilb, Tom, *Quantifying the Qualitative*, available at http://www.result-planning.com