



The Menlo Institute LLC

# Name That Function

How easily can humans decode your software?

## The Function Naming Game

(Notice how miscommunication has undesired effects.)

Player 1: "I can name that Function in fifteen characters."

Player 2: "I can name that Function in ten characters."

Player 1: "I can name that Function in three characters, with no vowels."

Host: "Name that Function."

Player 1: "dst"

Host: "Great, I take it you mean 'destination'?"

Player 1: "Well, actually I was thinking of 'destroy'."

**Thomas Meloche © 2002  
Fellow, The Menlo Institute**

**The Menlo Institute LLC  
212 N. Fourth Ave  
Ann Arbor, MI 48104  
[www.menloinstitute.com](http://www.menloinstitute.com)  
(734) 665-1847**

### ***Mars or Bust***

Early in the morning of September 23, 1999 the Mars Climate Orbiter fired its main engine to go into orbit around the planet Mars. Up to that point, all of the information coming from the spacecraft looked normal. As seen from Earth, the engine burn began as planned five minutes before the spacecraft passed behind the planet. However, flight controllers did not detect a signal regarding when the spacecraft was expected to come out from behind the planet. The Orbiter was missing.

Seven days later, the Mars Orbiter Team issued a press release. The press release identified three contacts, one located at Headquarters, in Washington D.C., the next at the Jet Propulsion Laboratory, in Pasadena, California and the third at Lockheed Martin Astronautics in Denver, Colorado. It is likely that you already know the story. According to a preliminary peer review one team used English units (e.g., inches, feet and pounds) while the other used metric units. They used different units when communicating key spacecraft information. As a result the spacecraft could not be put in proper orbit. The Orbiter was lost.

Dr. Edward Stone, director of the Jet Propulsion Laboratory stated that they already had a thorough investigation of the incident underway. Two separate review committees were formed and I am sure that there were many meetings. An independent NASA failure review board was also formed. As Dr. Stone stated their inability to recognize and correct this simple error had major implications. The Orbiter was destroyed.

### ***Speculation***

At the time this paper was originally written, no additional details had come forth other than those stated above. Clearly there were significant problems in the development process. The failure listed is especially irritating because almost everyone in the business knows of so many different ways to prevent it; none of them particularly expensive or difficult. They certainly aren't rocket science.

The Orbiter problem got me thinking of similar types of problems I encounter on almost every project I ever visit - poor interfaces. I am not talking about the graphical user interface, although it is usually poor too. I am referring to system level interfaces, component interfaces, object interfaces and function call interfaces; especially interfaces between groups of engineers.

Developers think they are developing software for machines. In fact, they are developing software for the other people on their teams including fellow developers, maintenance and support engineers.

**Example**

Here is a fictional example of what I mean. Imagine we are writing spacecraft software (hmm - now how did I get that idea?) and somewhere in our fictional spacecraft an interface exists with a method of SetShipAltitude.

```
SpaceCraft { ... SetShipAltitude ( int height ); ... };
```

This interface, again a fictional example, looks alot like interfaces I see on software everyday. The routine named SetShipAltitude does not provide sufficient information about what the height parameter is, is it in inches, feet, meters, furlongs? To answer this question, additional documentation (or the code itself) must be read.

The problem is escalated if the function SetShipAltitude already exists in other parts of the system. Especially if every other SetShipAltitude already accepts a parameter of meters. It is easy to assume that this one is in meters too. Of course, in this one case that would be wrong, since it is in feet. Too bad, the space craft will be lost. The developer should have read the comments embedded somewhere in the source code. Or perhaps they should have read the 200 pages of supplementary documentation - if they were ever written and if they were up-to-date perhaps they would help.

***Name That Function***

Software developers must consider typing speed to be the limiting factor in why software is delivered slowly. Only that explains why they constantly write in shorthand and leave out vowels. Only that explains why they publish guidelines to each other that say ridiculous things like, no function or method name should be over 15 characters long.

Now, if typing speed is the limiting factor on your ability to deliver quality software then by all means address that problem, by giving your developers typing classes and typing tests. But if typing is not the limiting factor in your ability to produce and maintain robust quality software then perhaps developers should type more characters and make the code simple to read and easy to understand.

***Here is a secret:***

Teach the developers to write software for other developers, not for the machine. Other developers are the most important customers for readable code. The machines have no problem reading code, the developers do.

The machines will do exactly what you tell them to do. If you tell them to fly

into Mars they will fly into Mars. They are very accommodating. The developers will do what they think they should do, which may or may not be what you want them to do.

Set up coding guidelines with the primary goal of clarity. Have the team create code so that it is clear to developers and simple to understand. The next developer coming along may come simply to call the code or to update and maintain it. In any case, they need to understand it and that understanding begins at the interface.

An example of an improved interface in our mythical space craft could be:

```
SpaceCraft { ... SetShipAltitudeInMeters ( int height ); ... };
```

Here, at least, it is made clear in the interface that the altitude is in meters. It is unlikely that a consumer of this service would consider it to be feet. Of course, you do have to type 23 characters. But how does the cost of typing 9 extra characters compare to crashing a spacecraft? An even better interface in an object-oriented system may be:

```
SpaceCraft { ... SetShipAltitude ( LinearMeasure height ); ... }
```

Here `LinearMeasure` is an object that can take and translate measures in English units or Metric units. It will do the appropriate translation for you. Assuming we are using Extreme Programming unit tests, and have established a disciplined process where we have built a rigorous set of unit tests, our `LinearMeasure` class should not fail, and we should not have to worry about specific measurement units at all.

It is interesting to note that the probable cause of the loss of the Orbiter was published within seven days of the loss. It is often the case that it is relatively easy to find a mission critical failure point once you know it exists.

The Orbiter failure serves as yet another example that we need to pay attention to our processes, specifically how we communicate. We clearly have room for improvement. Lets start with an easy fix to implement, how we “Name That Function.”

**About the Menlo Institute**  
 Founded in 2001 by Thomas Meloche, Richard Sheridan, James Goebel and Robert Simms, The Menlo Institute specializes in Software Development Process and Methodology with a special focus on an agile software development environment called The Software Factory.

**The Software Factory**  
 The Software Factory is a complete agile software development methodology created at Menlo. The Software Factory combines best practices from Alan Cooper's Interaction Design, Kent Beck's Extreme Programming, The Rational Unified Process and Six Sigma. The Menlo Institute has unique insight from real-world experience on how these practices can work together effectively.



**Thomas Meloche  
 President  
 The Menlo Institute**

Thomas Meloche is President of The Menlo Institute LLC and a founding member of Menlo Associates LLC. His passion is software development best practices and building successful software teams. Before founding The Menlo Institute, Thomas was the Director of Engineering at AppNet, Inc. For over seven years, he led a consulting staff that grew from

30 people to over 240 people which generated revenue of over 100 million dollars. As a result, he is intimately familiar with the real problems that befall real software development projects. He also knows how to rectify them. During his tenure his delivery organization had a remarkable 100% successful delivery record.

Thomas has a B.C.E. in Computer Engineering from the University of Michigan, and especially enjoys providing training back to his alma mater. He has taught classes and provided mentoring for both the University of Michigan Law and Medical Schools.

Menlo Institute Classes by Category		
Extreme Programming (XP)	Foundation Courses	Rational Unified Process (RUP)
Introduction to XP Coaching XP Teams Writing Story Cards XP BOOT Camp / Immersion	Secrets of Software Success Six Sigma Software Building a Software Factory Object Technology Overview	Managing with RUP-Lite Capturing Reqs With Use Cases Practical UML RUP BOOT Camp / Immersion
Requirements Gathering	Object Technology	Project Management
Capturing Reqs With Use Cases Writing Story Cards	Object Technology Overview Obj. Oriented Analysis & Design Advanced Obj. Oriented Design	Managing with RUP-Lite Software Project Mgmt Six Sigma Software