# The API Walkthrough Method

## A lightweight method for getting early feedback about an API

Portia O'Callaghan
MathWorks
portia.o.callaghan@mathworks.com

## Abstract

We propose a method for evaluating the usability of an Application Programming Interface (API) in the context of MATLAB, a high-level programming language. The primary goal is to evaluate whether the participant can develop an accurate mental model of the API based on the code alone. Like traditional usability testing, this method takes place in a lab setting with a facilitator and observers, and a single participant is exposed to a prototype. Unlike traditional usability testing, the prototype is a static text document containing a series of programmatic statements. Rather than performing a task, the participant "walks through" the code line by line in an attempt to gain understanding of the system. Using standard usability testing protocols, the facilitators are able to assess whether the participant understands the API, as well as gather preference data between two designs.

***Categories and Subject Descriptors*** G.m [*Software*]: Miscellaneous - Software psychology.

***General Terms*** Design, Human Factors, Languages.

***Keywords*** usability; usability testing; API usability; programming languages; API; API evaluation; programming interface

## 1. Introduction

At MathWorks, design is a collaborative activity. APIs are typically designed by a small team that includes one or more developers, a quality engineer, a usability specialist, and a documentation writer. We believe that cross-functional team collaboration is a key factor in the effectivenes of our design process.

We follow a short six-month release cycle, making it critical to obtain usability feedback on our designs as early in the release cycle as possible. Early feedback enables us to fine-tune an API, or even to change course before the API is implemented. After implementation, changes become more expensive, and the API's underlying model has been established in everyone's minds. As a result, teams are very hesitant to make extensive changes to the API after implementation.

The Usability Department at MathWorks employs many lightweight[1] methods for getting usability feedback early in the release cycle. Paper prototyping [8] is commonly used for graphical user interfaces, but we hoped to find a method that could be applied to APIs.

Traditional methods have been used to evaluate APIs [1], but these methods suffer from various limitations for early-cycle testing. Traditional usability testing, where the participant attempts to use a prototype to accomplish a task, is often not possible early in the release cycle because the API has not yet been implemented[2]. Surveys are good for establishing trends and getting preference data, but they don't allow us to explore the workflows that the API enables as well as lab-based one-on-one usability sessions.

Some methods provide excellent results, but they may be impractical in a fast-paced, resource-strapped industry setting. The Cognitive Dimensions framework has been applied to API design [2], but this method requires significant training and buy-in from the design team, which may not be practical. Instant-messaging API testing [3], which uses an IM client to present the computer's responses to command-line statements, is a good alternative for early-cycle usability testing. However, this type of test requires someone to play the role of Computer at every usability session, and sometimes this is impractical.

## 2. Background

### 2.1 About MathWorks and MATLAB

MathWorks is the leading global provider of software for technical computing. The company's flagship product is



[1] *Lightweight* in this context means getting results with minimum effort.

[2] Although it normally doesn't take much effort to create a prototype API for testing purposes, we often find that the right people aren't available to implement a prototype at the time we need to test the API.

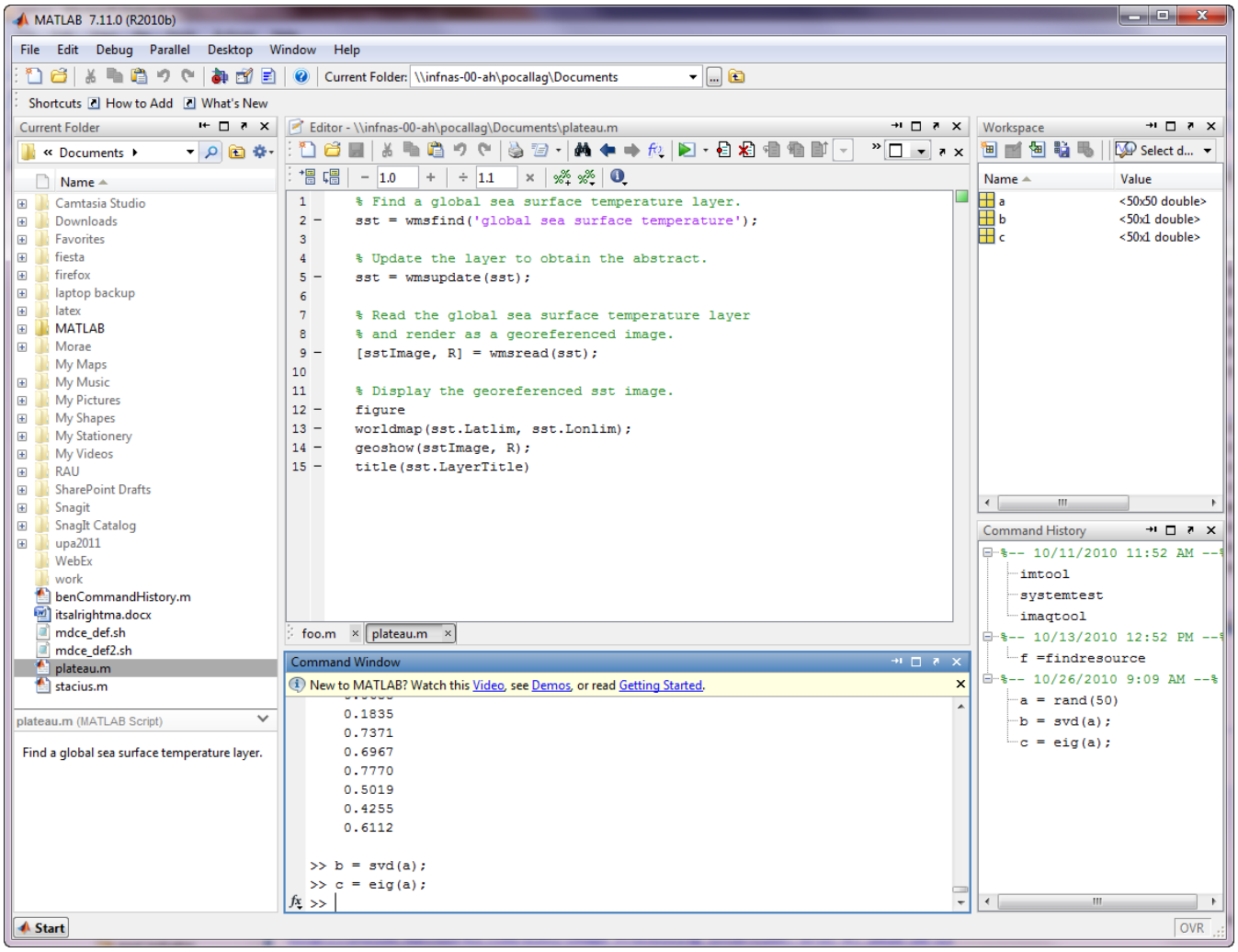


**Figure 1.** The MATLAB Desktop

MATLAB; a high-level, untyped, interpreted programming language and environment that is optimized for rapid prototyping and is typically used by scientists and engineers. The beauty of the MATLAB language is that the user doesn't need to know much about programming; it's relatively easy to get started and to get results. When designing new functionality for MATLAB, it is therefore extremely important to facilitate understanding by novice users, even for complex functionality. For an API, this often means that the names for MATLAB functions, classes, properties, and methods must be easy to understand and require as little documentation as possible. This translates to increased productivity and reduced programming effort for all users.

MATLAB users can write MATLAB code to be interpreted at the command line, or they can write scripts or functions in the MATLAB Editor. The Editor is part of the MATLAB Desktop, which is the native IDE for working in the MATLAB environment. The Desktop also includes a Variable Editor, a Current Folder browser, a Command Window, a Command History window, and a Workspace Browser, as well as many other optional GUI features. See Figure 1.

Functional programming is the historical basis of MATLAB features, but more recently some features have been implemented in an object-oriented manner. As a result, some MATLAB users are becoming familiar with the use of objects, properties, and methods. However, since MATLAB users are not typically computer scientists, it is critical to provide usable object-oriented APIs.

### 2.2 Mental Models

Early in the release cycle, we often want to evaluate whether the mental model [4] of an API is accurate or not. An *accurate mental model* is achieved when the user's mental model is consistent with the underlying system model.

An example of an accurate mental model is as follows. In MATLAB we have an API for communicating with schedulers that send computational work to computer clusters. Let's say we represent the scheduler with an object called

Scheduler. When instantiating a Scheduler object, the user may wonder whether they are creating a new scheduler process or creating a connection to an existing scheduler. To remove this ambiguity, we could call this object a SchedulerConnection or a SchedulerProcess. Each of these object names conveys a different mental model to the participant. It's important that we choose the right model and design the API accordingly, so that the participant can effectively work with the API to use the external hardware. With the API Walkthrough method, we can try out a few different names for a class, and observe whether the participant understands the system or not. If she does, we can say that our API conveys an accurate mental model.

## 3. Goals

Our goal is to develop a lightweight method that can be used to evaluate a programming interface at an early point in the release cycle, before the API has been implemented. Specifically we aim to evaluate:

- whether the participants can construct an accurate mental model of the system just by "walking through" some code samples.
- whether the input and output formats and/or classes make sense to the participants.
- whether the names of the functions, classes, methods and/or properties give participants a clear understanding of the system.
- what makes sense and doesn't make sense to the participants, so that our approach to the API design can be changed if necessary.
- the readability of the code.

To achieve these goals, we propose a methodology which utilizes existing artifacts to provide a context with which participants can evaluate a new API design. During the study, the facilitator will ask the participant to walk through a script of code and explain each step.

## 4. A Simple Case Study: Web Map Server

A few years ago, we developed a new API that would allow users to find imagery on the web from Web Mapping Servers (WMS) and bring it into MATLAB for further analysis. We wanted to find out whether the new API would make sense to users, and more specifically, whether the functions we designed would help users arrive at an accurate mental model of our underlying system so that they could work effectively with the API. In addition, we wanted to test two different options for some of the function names with real users, to find out which option helped them come to an accurate mental model, as well as which one they preferred.

We started by identifying use cases. One important use case was to find recent imagery for sea surface temperature, and to plot it against a map of the world in MATLAB. Based on our API design and our use case, we came up with the design case[3] in Figure 2.

```
% Find a global sea surface temperature layer.
sst = wmsfind('global sea surface temperature');

% Update the layer to obtain the abstract.
sst = wmsupdate(sst);

% Read the global sea surface temperature layer
% and render as a georeferenced image.
[sstImage, R] = wmsread(sst);

% Display the georeferenced sst image.
figure
worldmap(sst.Latlim, sst.Lonlim);
geoshow(sstImage, R);
title(sst.LayerTitle)
```

**Figure 2.** A simple design case

We then removed the explanatory comments and put the code into a MATLAB code script as in Figure 3.

We also created an identical code script which presented the same code, but with different function names, such as *wmslocate* rather than *wmsfind*. The purpose of having two code scripts was to see whether users understood the underlying system more effectively with one function name versus another. Would 'wmsfind' give users the erroneous mental model that MATLAB was searching the internet for imagery? Would 'wmslocate' do a better job of conveying the correct mental model – that we were in fact searcing a local database that shipped with MATLAB? These were some of the questions we hoped to answer.

We presented these code scripts to participants in random order, so that some participants saw script A first, and some saw script B first, but all participants saw both scripts. By asking the participants to walk through the first code script and tell us what was happening on each line, we were able to fine tune the function names and signatures, as well as the documentation.

```
sst = wmsfind('global sea surface temperature');
sst = wmsupdate(sst);
[sstImage, R] = wmsread(sst);
figure
worldmap(sst.Latlim, sst.Lonlim);
geoshow(sstImage, R);
title(sst.LayerTitle)
```

**Figure 3.** Example code script

After each participant saw the first code script, we told them that we had another very similar script and we wanted to see if the second script changed their understanding of what the code did, as well as which function names they

[3] A design case is a "solved use case" – see Section 5.1.

preferred. When the participants walked through the second code script, they typically did not spend as much time on each line, but instead skimmed the script for changes. We recorded comments such as, "Oh, I thought that function was looking on my hard drive, but now that I see this other name, 'wmsfind,' it gives me the impression that it's looking on the internet for imagery."

This simple case study demonstrates that the API Walkthrough method can be used to find better names for functions. However, more fundamental types of changes can be made to an API as a result of this method (see Section 6).

## 5. Methodology

### 5.1 Preparation

Before designing the study, the team should have two kinds of source material. First, a set of documented use cases that describe the primary workflows that the API is intended to enable. At MathWorks, use cases are written in the context of the current functionality in order to expose the pains that current users encounter. In some cases, the use case describes an intended goal, but does not specify a solution if there is no way to accomplish the task with the current system. Second, the team should have an idea of what the new API will be, ideally in the form of a specification that defines at least the basic outline of the new API.

From these, a set of *design cases* (see Figure 2) is created; these show, in code, how the user would accomplish these common tasks using the new API. Design cases demonstrate an envisioned future use of the system, based on the functionality yet to be implemented. They help developers and usability specialists consider the planned use of a new tool.

Based on the design cases, a text file or *code script* (see Figure 3) is created which shows the commands that the user would enter to accomplish common tasks. The code script provides the participant with the steps of a completed task; the facilitator will ask the participant to walk through the task and explain each step. It is important that the code script not contain any explanatory code comments, and that the variable names are generic enough so as not to give the participants too many clues about the meaning of the API.

The code script may contain some task-related comments, such as "You go home for the day and close MATLAB. The next day, you restart MATLAB and evaluate the following commands." This comment would be useful if for example you want to evaluate whether participants understand which objects persist after a restart, and what state those objects are in.

For our MATLAB code scripts, we separate the tasks into chunks[4]. This allows a natural end to each task and promotes conversations about each task separately. For the study observers, you may create an annotated version of the code script, with more commentary about the inputs and outputs of each function call, open questions, etc.

[4] We use cells in the MATLAB Editor to display the chunks.

### 5.2 Recruiting

Recruiting for an API walkthrough study is no different than for any other usability study. Real users are recruited, i.e. customers whose skills and experience match the profile of the expected or target user. The profile is defined as part of our cross-functional design process and is documented in our use cases. The profile typically describes a casual or experienced MATLAB user who is versed in the domain area (e.g. image processing, controls engineering) and who has some expertise in programming in the MATLAB language. When the functionality caters to a more seasoned software engineer, that is reflected in the target profile for the feature. Regardless of the functionality we are designing, we typically recruit users from across the spectrum of experience levels.

### 5.3 Execution

Before starting the study, we tell the participant that the API that they will see has not been implemented yet, and therefore they will not be able to execute any of the commands in the code script. Part of the purpose of the study is for the facilitator to see what does and doesn't make sense for the participants, so that we can rethink our approach to the API design if necessary. We also tell participants that there is no documentation available, and that if they feel confused by the code, to note that and move on with the interpretation of the code script.

We present the code script in the context of MATLAB in the MATLAB Editor. Although the participant cannot interact with the commands or variables, they can see the commands with the familiar code highlighting and colorization that comes for free with the MATLAB Editor.

After answering the participant's questions and reminding them to use the think-aloud protocol [6], the facilitator asks the participant to start verbally "walking" through the code.

### 5.4 Facilitation

The two most critical aspects of facilitation with this method are making the participant feel comfortable and qualified to give feedback, and making sure the participant gives a running commentary of their thoughts.

We use the think-aloud protocol [6] to ensure that we hear what participants are thinking as they are walking through the code. We want to hear what participants think the outputs of a command are, the classes of variables, and the specific behavior of each function.

More than in other types of usability activities, the participant is relatively blind about what the code is doing. They are unable to run commands or inspect variables because the code has not yet been implemented. As a result, they of-

ten experience additional frustration. The facilitator must be sensitive to the potential for these effects [8].

Additionally, participants sometimes state that they cannot continue without seeing some documentation. Since there is no documentation, the facilitator needs to provide help as necessary, but without giving too much help at once. The Incredibly Intelligent Help method can be of use here [7].

### 5.5 Debrief

After the usability studies, we use similar debrief methods to other studies. Ideally we schedule a separate debrief session with the team and any other observers of the session, where we discuss our individual findings and affinitize them into categories, then prioritize the findings [5].

### 5.6 Variations

#### 5.6.1 Code-writing tasks

We often ask the participant to write a code snippet based on the designed API after walking through the code script. A coding task is not intended to produce working code, but to reveal whether the participant has a basic understanding of the API. This part of the test can often confirm whether the participant has formed an erroneous or an accurate mental model.

#### 5.6.2 Preference data

Another variant of this method (see Section 4) involves presenting two code scripts in random order, each with different function or class/property/method names but with the same basic structure. Participants spend the bulk of their time with the first code script; the facilitator then asks them to review the other code script to see whether the different names provide a different mental model or a different understanding of the code.

#### 5.6.3 Testing the API alongside a GUI

A third variant involves presenting the API before or after a related GUI (either a paper prototype or a live prototype). Participants are asked to perform tasks with the GUI as in a typical usability test. The API Walkthrough may either validate the participant's understanding of the underlying system as seen in the GUI, or it may produce confusion. The design team might wish to make changes to the GUI and/or the API, depending on the findings of the test.

## 6. Results

We have used this method for the past 18 months with four API design projects. In general we have had great success with this method, with each study having one or more of the following outcomes:

- We evaluated whether the underlying system that is exposed by the API makes sense to participants. When we found that participants were confused by the designed API, we made changes to the design and repeated the test until we saw a successful outcome.
- We evaluated which function/object/property/method names make the API easier to understand.
- We evaluated how well a given API and a corresponding GUI work together.
- We used the results to convince stakeholders of the efficiency of our design choices.
- We found many opportunities for enhancing the documentation of an API - usually before the documentation was written.

In one recent study, we discovered that the designed API encouraged a critical misunderstanding in the participants' mental models. Our design combined two key concepts into a single class, which we had hoped would simplify the API. Using the API Walkthrough method, we discovered that this simplification confused users. We were able to make changes to the API and repeat the test, and we found that the new API did not produce the erroneous mental models. After another study on an image processing feature, we changed the class of a function's output after we found that every participant in the study assumed that the function would output a simple nx2 matrix of X-Y coordinates rather than a more complex data structure.

In addition to the simple naming decisions demonstrated in Section 4, there are other types of API design decisions that can be made as a result of the API Walkthrough method, such as:

- Whether to combine functionality into one function or separate functions
- What the default settings of a function or class should be
- How much additional functionality to provide with non-default parameters
- Whether to expose functionality in functions or classes
- What format/class to specify for function inputs/outputs

### 6.1 Benefits

Some benefits of using this method that distinguish it from other methods include:

- It can be used very early in the design cycle, before any code is implemented.
- It can be used before any documentation is written.
- It is a natural part of the design process at MathWorks, since we already create use cases and design cases.
- It can be used in design reviews as well as in a usability testing context.
- It can be combined with a GUI evaluation method, if the API and the GUI would be used together.

### 6.2 Potential Weaknesses

Some potential weaknesses of this method:

- Participants may feel more anxiety than in other usability studies. With a GUI, people may feel more free to give feedback. The facilitator must take special care to address this.
- Some participants will not feel comfortable guessing at what the API does - they may repeatedly ask for documentation. This is usually addressed by the facilitator: "Since we haven't implemented this yet, there is no documentation. We are at a really early phase of the development and we really want to know what people expect it to do. What would you expect it to do at this line?"
- There is a danger that participants will misinterpret the API in a way that isn't clear to the facilitator; this may be because the participant can't validate their assumptions by interacting with the API. A code-writing task can ameliorate this.

## 7. Future Work

The API Walkthrough method has been useful in designing APIs at MathWorks. We intend to continue fine tuning this method, as well as to evaluate the feasibility of applying it to more general design patterns [9] in MATLAB and other programming languages.

## Acknowledgments

Thanks to Will Schroeder, Amy Kidd, Alex Feinman and Rachel Cobleigh for their thoughtful feedback and encouragement. Thanks also to Kelly Leutkemeyer for his work on the WMS functionality in MATLAB.

## References

[1] J. Beaton et al. *Usability Evaluation for Enterprise SOA APIs.* Proceedings of the International Workshop on Systems Development in SOA Environments, 29–34, 2008.

[2] S. Clarke. *Measuring API Usability.* Available at `http://www.drdobbs.com/184405654`, 2004.

[3] R. L. Cobleigh and D. Cooper. *Using instant messaging to usability test an API.* Proceedings of the Usability Professionals' Association International Conference, June 2008.

[4] D. A. Norman *The Design of Everyday Things.* Doubleday, 1988, 16–17.

[5] M.B. Rettger. *Post-its™ and Affinities: Low-tech tools for high-impact results.* Presented at the Usability Professionals' Association International Conference, Austin, TX, June 2007.

[6] J. Rubin. *Handbook of Usability Testing.* John Wiley & Sons, 1994, 217–219.

[7] T. Scanlon. *Six Slick Tests for Docs and Help.* Available at `http://www.uie.com/articles/documentation_help/`, 1998.

[8] C. Snyder. *Paper Prototyping.* Morgan Kaufman, 2003, 177–186.

[9] J. Stylos et al. *Comparing API Design Choices with Usability Studies: A Case Study and Future Directions.* Proceedings of the 18th Workshop of the Psychology of Programming Interest Group, 131–139, 2006.