1 Introduction

Scientists construct quantitative models to explain observations about natural systems in a coherent and rigorous manner. These models can be characterized by their scope and validity. The scope of a model is the set of observable quantities that the model can generate predictions about, and the validity of a model is the extent to which these predictions agree with empirical observations of those quantities. Today, as the number of models and the quantity of empirical data increases, scientists face a grand challenge: efficiently discovering relevant models and characterizing their validity against a continually growing body of available evidence. A scientist typically proposes a new model by publishing a description of how it works along with an argument justifying its utility to some targeted scientific community. This argument is made in words, accompanied by a few relevant equations and figures, and judged by the process of scientific peer review. Ideally, reviewers determine whether the model's predictions are consistent with all available and relevant data and compare their accuracy against those produced by previously published models. This is increasingly difficult; although authors are expected to facilitate this process by referring to relevant experimental data and conducting a literature review, these citations are likely to be both incomplete and biased, in that authors will dedicate the most space in their publications to data that their model explains well relative to a subset of previously developed models, or to an implausibly simple null hypothesis.

If each modeling paper contained figures a) highlighting the model's accordance with every related experiment and b) comparing its performance to every other related model, then publications would be encyclopedic and their main points would become obscured. The strength of model publication as it stands is the focused description of how a model works and its conceptual and technical advances. A weakness, however, is that evaluating the scope and validity of a published model is intractable using the contents of the publication alone. In other words, publications tell us clearly how a model works but provide only incomplete outlines of which goals it hopes to achieve and how well it achieves them. This problem is only exacerbated as more data is gathered in the years following the original publication. Although the validity of a model may change in light of new experimental observations, there is no systematic process in biology today for re-evaluating existing models. While new data and its most important theoretical implications may propagate informally through a scientific community or appear in periodic reviews, the original publications, which serve as a resource of first resort for new scientists and others outside of the community, will be cited "as-is" in perpetuity.

We can thus distill the central problem we seek to address in this proposal: the process by which models are validated is not sufficiently rigorous, comprehensive or ongoing. This makes it difficult for scientists to identify which models best predict quantities of interest, to compare proposed models against one another, and to precisely identify problems that have not yet been adequately resolved in their research area. To overcome these problems, we propose systematizing the model validation process by creating a collection of software and associated cyberinfrastructure in support of a more systematic scientific model validation process. This model validation framework will exist in parallel to the publication system, allowing the latter to focus on answering "how", while referring to the former for a comprehensive answer to "how well".

1.1 Existing Efforts

While there have been several facilities developed for data and model sharing in biology, there have been few attempts to facilitate evaluation of models *against* data. For example, the Collaborative Research In Computational Neuroscience (CRCNS) data-sharing website (http://crcns.org/), and the Open Source Brain repository (http://opensourcebrain.org/) are separate facilities for data and model sharing in neuroscience,

respectively. The CRCNS website is specifically focused on data sharing for the benefit of computational modelers, and has benefited from community contribution of several excellent data sets, but it does not explicitly relate available data to models in any way. The latter resource, the Open Source Brain Initiative, is focused on model description and execution, and is an emerging example of the power of standards in informatics. Models are published as-is, without any information about which specific data the published models aim to predict and how well they achieve their goals. The work proposed here aims to bridge these two kinds of resources, strengthening each in turn.

There are related efforts in the machine learning community to develop models and validate them against standardized, publicly-available datasets according to well-specified metrics. This mechanism underlies the Kaggle website (http://kaggle.com/), where groups seeking machine learning models of their data can organize competitions where developers receive a set of training data and submit competing algorithms that are compared automatically by their cross-validated accuracy on an unrevealed test set. The success of Kaggle [ref?] shows that open competitions are highly effective, and that this paradigm of "modeling as a competition" draws in large numbers of data scientists across traditional discipline boundaries. However, the models developed in this way must fit into a few very general use cases in machine learning: classification, regression, and clustering. The validation criteria are relatively straightforward in these domains. More specialized or open-ended modeling challenges do not fit well into the Kaggle model today.

Discipline-specific competitions in biology have also resulted in technical advances for specific problems. For example, the quantitative single neuron modeling competition [**?**] has helped us understand the complexity-accuracy tradeoff among reduced models of excitable membranes, and identified several specific models with the best overall predictive power over a range of inputs. Another competition, the "Hopfield challenge" [**?**], famously illustrated through its difficulty the challenges facing computational neuroscience. [[Other examples from biology still needed.]] Our challenge is to develop a general framework in support of such distributed data-driven model validation workflows, where the validation criteria may be complex and discipline-specific. We will be particularly focusing on validation challenges in the biological sciences, and specifically neuroscience, in the work proposed here, but anticipate that this work will be applied to other scientific areas in the future.

2 Outcomes and Products

Our framework is centered around simple *validation tests* that compute the agreement between a result from a computational model's execution and a single experimental observation. The overall validity of a model is then identified as the collection of validation tests that the model passes. This methodology is inspired directly by the nearly ubiquitous practice of *unit testing* in software engineering. A *unit test* evaluates whether a portion of a computer program (often a single function) meets one simple correctness criterion. Due to the narrow scope of each test, failed tests precisely identify parts of a program that are not functioning correctly. A *suite* of unit tests that collectively *cover* the desired behaviors of a program serve as evidence that, overall, the program being tested is functioning as desired.

Developers often write unit tests before writing the program itself, following a methodology known as *test-driven development (TDD)* [**?**]. TDD is based on the idea that a suite of unit tests can itself serve as a program's specification and guide its development. Following the TDD methodology allows developers to measure development progress simply by looking at the proportion of tests that pass at any time during development. Once a program has matured, developers continue to use the test suite to ensure that changes did not break existing functionality by checking that the tests that previously passed continue to do so following the change. The success of unit testing and TDD across diverse application domains in software

engineering suggests that validation testing may also provide foundations for a practical solution to the model validation problems discussed above.

2.1 Example: Neural Membrane Potential Dynamics

To make our proposal concrete, we will begin by describing an example test related to neuroscience. Our test is derived from experiments where a stimulus was delivered to neurons of a particular type, in the form of somatically injected current (reported in pA), while the somatic membrane potential of each stimulated cell (reported in mV) was recorded and stored. A model that claims to have captured the dynamics of this cell type's membrane potential can be validated in several ways, but one simple test (among many) of that claim would check for correspondence between the distribution of the number of action potentials produced by the model and the distribution observed across cells in the data. This test would ask any candidate model being tested to produce a prediction for each injected input current and check whether the predictions fall within the empirically-observed distribution of action potential counts seen in response to that input current. The specific metric for success would be specified by the test creator. For example, models that, for each input current, produce action potential counts that fall with the 95% confidence interval of the empirical data distribution might pass our example test. Importantly, this choice is made explicit in the specification of the test, so modelers need not guess which criteria potential reviewers were used to validate their model.

2.2 SciUnit: A Simple Validation Testing Framework

We consider conceptual and practical simplicity – the absence of "heavy-lifting" on the part of practicing scientists – to be an essential requirement for community participation. Our design, by abstracting data behind tests, does not require the storage or release of raw data in standardized formats. Instead, only the salient aspects of the data (such as the calculated action potential count distribution in the example above) are needed to create a test. Similarly, the implementation details of a model (e.g. its programming language) need not be public or standardized – the implementation must simply be wrapped to expose a standardized interface, so that tests can access necessary model capabilities. Each community can collaboratively specify model interfaces that capture the high-level operations of their preferred modeling formalisms. Existing modeling standards (e.g. NeuroML [**?**]) and interface definition languages (IDLs) [**?**] will be leveraged to simplify this process. **Based on these criteria, the first product of this proposal is a simple, general object-oriented validation testing framework called SciUnit, written in Python, that makes the construction (from data) and capability-based model execution as easy and flexible as possible.** We will initially focus on multiscale neuroscience models by developing NeuroUnit as an application of SciUnit (see Section 2.5 below), and other fields of biology will similarly be based on this same core validation testing framework in the future.

2.3 Community Workflow

Unit tests are generally produced by the community of developers and architects involved in the design and development of a particular program. In our proposed framework, members of a scientific community will produce suites of validation tests collaboratively in common source code repositories. Each test will specify the requirements for candidate models, and provide a procedure for determining whether a candidate model (the input to the test) is consistent with a particular summary of experimental observations (the parameters of the test). The performance of a model on a collaboratively curated suite of such validation tests can then serve as justification of claims regarding the validity of that model as a description of the scientific

system modeled by the test suite. This workflow will continuously produce, on behalf of the community, a summary of the field indicating which models are capable of taking each test, i.e. whether the test falls within the models scope, and for each capable model, how well it performed, i.e. its validity. The overall usefulness of a model is then a function of the weight given to each test, determined either individually by each investigator or by community consensus. This test-driven scientific workflow leverages the natural desire of modelers to promote the virtues of their models – suites of tests represent clear challenges, issued by experimentalists to the modeling community, and passed tests unambiguously certify success for the modelers. As more data is gathered to refine test suites, the collection of previously developed models can be tested against the most up-to-date empirical view of the field, often in a fully-automated manner because the interface between tests and models will remain fixed in most cases (only the empirical parameters will change).

2.4 SciDash: A Community Web Portal

The value of this framework is highest when the full and current state of a research area is represented by a coherent collection of tests and models. This requires coordination among the research groups in a research community, so community-oriented cyberinfrastructure to support test suite creation and summarization, building upon SciUnit as the underlying testing framework, is also an essential component of our proposal. This service, called SciDash, will utilize existing infrastructure for coordinated development of software repositories, initially focusing on GitHub (http://github.com) [[ref of github in science?]]. A suite of related tests will be contained in a GitHub repository with a stereotyped high-level structure that allows the SciDash web portal to discover, index and summarize it on an ongoing basis. The portal, a web application written using the pythonic Django framework (http://www.djangoproject.com) will serve as a central location where scientists can discover relevant test suites, determine each tests requirements, and summarize the results of executing these tests on submitted models. With sufficient community contributions, test results will be visualized as a record matrix composed of large numbers of model/test combinations (Figure 2 provides a toy example). Each row in the record matrix will contain the results for all tests taken by one model and would serve as easy-to-interpret evidence of that models validity (or lack thereof). The models and tests themselves will be available for perusal via hyperlinks from the record matrix. SciDash will serve as a public record of competition among models, which should induce a more deliberate effort from modelers to build their models in accordance with data. These repositories will also serve a general organizational role in research communities that adopt them by encouraging the development of standards for both code and data. Thus, the second product of this proposal is a web application called SciDash, whose back-end can index and summarize test suite repositories being developed on GitHub, and whose front-end offers users a well-documented, easy-to-use environment for discovering and evaluating each collection of community-generated tests, models and test results.

2.5 NeuroUnit: A Suite of Tests for Neurophysiology

In order to demonstrate the utility of this framework and infrastructure, a substantial initial case study must be conducted. Because the investigators leading this proposal have substantial experimental and computational neurophysiology training and expertise, the work will initially be targeted toward this domain. By using machine-readable models from the Open Source Brain repository, and corresponding machine-readable data from resources like The NeuroElectro Project (http://www.neuroelectro.org), our efforts will produce a body of useful tests that also serve as a demonstration of the test-driven scientific workflow that

other areas in the biological sciences (and eventually other sciences) can leverage. The third product of this proposal is NeuroUnit, a large suite of data-driven tests and corresponding test results for a representative set of canonical neurophysiology models, all publicly available. A case study examining the challenges and practicalities of this process will be a result of this effort, and scientists in other biology subfields can use it to inform development of their own domain specific test-suites. .

3 Preliminary Activities

We have developed the core testing framework, SciUnit. It is being developed under a free, open-source license from the project development repository at http://github.com/cyrus-/sciunit.

3.1 SciUnit Implementation

We chose to implement SciUnit using the Python programming language (http://www.python.org) due to its widespread adoption across the quantitative sciences and its useful scientific computing facilities and useful object-oriented features. Python was also chosen because it supports interoperability with a number of other major languages used within science, including C [[ref]], R [[ref]] and MATLAB [[ref]]. The functionality described below could be readily translated into any programming language with comparable facilities, however.

3.2 Validation Tests

Testing in SciUnit proceeds in three phases (Figure 1): a) Capability Checking: The test checks to see that the model is capable of taking the test; i.e., that it exposes the needed functionality for the execution phase to proceed. b) Test Execution: The model is executed to produce the model output by interacting with its supported capabilities. If the model is described according to a standard specification, e.g. NeuroML, this corresponds to loading it into a supported simulator and executing it. c) Output Validation: The model output is compared to empirical data to determine whether the model passes. This comparison can be of several types: for deterministic models, we may extract a matching quantile from a data distribution; for stochastic models, we may compute a measure of agreement between model output and data distributions, e.g. the Kolmogorov-Smirnov statistic. Validation is ultimately pass/fail, but in some cases the result generated in this phase also includes a continuous statistic so that models can be ranked based on their test results in a more fine-grained manner.

A validation test in SciUnit is an instance of a Python class implementing the sciunit.Test interface. Classes that implement this interface must contain a `run_test` method that receives a model as input and produces a normalized score as output. The capabilities that the model must possess in order for the test to operate correctly are given using the `required_capabilities` attribute. In many cases, several tests will have a similar form, differing only by a choice of parameters or by association with a particular dataset. Such parameterized test families correspond to subclasses of sciunit.Test that take the needed parameters as constructor arguments.

In the example below, the RateTest class implements a parameterized family of tests that compare the output firing rate (action potentials per second) of a neuronal model, driven by a given somatic input current, to an empirically measured mean and standard deviation of the rate derived from a set of replicated experiments. The test produces a boolean score indicating agreement within one standard deviation of the empirical mean. The input currents, mean, and standard deviation are provided as constructor arguments

on line 2 (constructors are named `__init__` in Python.) On lines 5-8, the required capabilities are listed: the model must be able to take a current as input and produce firing rates as output. These capabilities are used in the `run_test` method, on lines 11 and 12 respectively, to execute the model against the provided current. The resulting firing rate is compared to the empirical mean and standard deviation on line 15 and a boolean score is produced on lines 16-20. In addition to the boolean value itself, the score also contains metadata that may be useful to scientists wishing to examine the result in detail. In this example, we save the models output rate and the provided mean and standard deviation alongside the result, by specifying the `related_data` parameter.

To create a validation test, a scientist instantiates the RateTest class with a particular mean, standard deviation and input current (I, not shown):

```
mitral_cell_rate_test = RateTest(mean=18.0, std=40.0, input_current=I)
```

The test is then executed against a model instance (described below), using the sciunit.run function:

```
score = sciunit.run(mitral_cell_rate_test, synchrony_model)
```

The sciunit.run function operates in three phases, implementing the abstractions at the beginning of section 3: (1) Capability Checking: The model is verified as capable of taking the test by checking each capability in the tests `required_capabilities` attribute. (2) Test Execution: The tests `run_test` method is called to execute the model and cache output. (3) Output Validation: `run_test` returns an instance of a sciunit.Score subclass containing a goodness-of-fit metric between the model output and the data used to instantiate the test.

Any errors that occur during this process are reported by raising an appropriate exception.

3.3 Test Suites

A test suite is a collection of tests designed to validate a model against several mutually coherent requirements.

```
urban_tests = sciunit.TestSuite([mitral_cell_rate_test, mitral_cell_interval_tes
```

When a test suite is executed against a model, it produces summary data that can be shown on the console or visualized by other tools, such as the web application described in the next section.

A test suite can also be used to compare multiple models (described below), producing a table like that shown below, or in Figure 2.

3.4 Candidates and Capabilities

A candidate is the entity to be tested. In practice, this will almost always provide an implementation for a particular scientific model, so candidate and model can be used interchangeably; we distinguish them allow for the possibility of testing datasets against each other as well. For clarity, we will refer to candidates as models herein. In our framework, a model is represented by an instance of a class inheriting from sciunit.Candidate, provided by the framework. In many cases, models can be grouped into parameterized model families, as with tests. These parameters correspond to parameters passed to a subclass constructor.

In order for a model to be tested, it must have certain capabilities that the test requires. That is, the model must be capable of taking certain kinds of input and generating certain kinds of output that the test will use to evaluate its quality. A capability is a contract that guarantees that a model is able to produce results of a particular form. We model capabilities as Python classes containing unimplemented methods. In other object-oriented languages, this would correspond to an interface or an abstract class.

Let us consider the capabilities of a model publicly available on the Open Source Brain website ([[URL for CA1 model]]). In the experiment being modeled, recorded data consists of somatic membrane potential

of a pyramidal neuron from area CA1 of the hippocampus, and the stimulus consists of somatically-injected current. This somatic membrane potential includes action potentials, the rate of which is known in neurophysiology as the firing rate. The following are some examples of corresponding model capabilities:

A wide variety of capabilities (as in example above) would be enumerated to span the modeling formalisms in use by a community. In the simple example we present here, these three capabilities suffice. As they are domain-specific (to neuroscience), we place them not in SciUnit itself but in NeuroUnit in the module neurounit.capabilities. In constrast, sciunit.capabilties contains only domain-independent model capabilities, e.g. ProducesNumber (does this model yield a number as output?), Runnable (can this model be run or is it static?)

A model is represented by an instance of a class inheriting from sciunit.Candidate. In the examples below, the CA1 cell model and Purkinje cell model families each exhibit different capabilities, which determine which tests are within their scope. These differences arise because the researchers who implemented the model did so at different levels of detail. In particular, the Purkinje cell was modeled with individual dendritic compartments, so there is an associated dendritic membrane potential.

In each example, the model produces a firing rate after being provided a current. This is implemented with multiple inheritance, so that each model expresses methods (such as `set_current`) that override (and implement) the abstract method provided in the corresponding capability classes. Any capability that the model inherits must have all if its associated methods implemented. Failure to do so will result in a NotImplementedError, which indicates that the model scope was less than what was claimed.

The model class is initialized with parameters (`i_leak` in the case of the CA1 cell model) to produce a model instance, which is essentially a model with specified parameters. A tester can specify that model instances taking a test should be initialized with particular parameters, or ranges of parameters. The writer of a test can provide as many or as few of these specifications as she feels are needed to instantiate models capable of passing the test. These specifcations may reflect important experiment context or metadata, such as known values of key system parameters, or the contents of a pharmacalogical milieu. In the RateTest above, the tester might specify the mean input resistance of recorded neurons; while this value may not be of interest in itself, it may be required to initialize a model instance capable of passing the test. The specification could also include a subset of the data of interest, which the model instance can use for training prior to a testing phase. Figure 1 illustrates the testing workflow, and Table 1 provides definitions for terms used here.

3.5 SciDash Development

We have also begun work on the SciDash web portal, which will serve to aggregate tests and their results for the scientific community. A development version is hosted at (http://www.scidash.org). We have populated it with the models and tests from the educational illustration in Table 2. This application has several key features. First, it will provide a dashboard view of the state of a field by displaying a matrix of records for selected model/test combinations. By designing the site with modern javascript-based data visualization libraries, users will be able to filter this matrix quickly to obtain the relevant subset of models and tests most relevant to their scientific questions. Second, each record will link to a Sage notebook worksheet displaying line-by-line the code that was executed and any intermediate output statements. Sage (Stein and Joyner, 2005; www.sagemath.org) is free open-source mathematics software heavily funded by NSF, based on a Python environment. The Sage notebook (www.sagenb.org) is a web application whose worksheets offer a means to publish and collaborate on a block of code, and to view its numerical and graphical outputs. While Sage was written for Python, the notebook also supports the execution of code and visualization of

outputs from other programming languages, making possible worksheets based on MATLAB, Mathematica, and other popular environments for modeling. Third, a community-moderated comment section will allow test-makers and test-takers to discuss issues associated with each record. Thus, disagreements about the appropriateness of a test can be openly aired and in many cases resolved. Rather than require special login credentials, we support open authentication via existing web portals (Google, Yahoo, Twitter, etc.), lowering the barrier to participation. The SciDash backend will consist of a MySQL relational database and XML files.

4 Research and Development Plan

In order to 1) accelerate development and testing of the framework and 2) populate it with models and tests, we will focus during the period of this grant on one scientific discipline (neurophysiology), without compromising generalizability to other fields. This will allow us to get rapid feedback from an experimental community the authors are familiar with as we iterate through the software development cycle.

4.1 Leverage of existing resources for the development of NeuroUnit

Here we describe standard tools in neuroinformatics that we have adopted to develop NeuroUnit (http://github.com/rgerkin/ne with the neurophysiology community in mind.

4.1.1 Candidates from NeuroML Models

NeuroML is a standardized model description language for neuroscience [[refs]]. It permits most neurophysiological and neuroanatomical models to be described in a simulation-independent fashion, and these models can be run in any of a number of popular simulators due to the programmatic inter-conversion capabilities of the NeuroML API. Because NeuroML is an XML specification, which can be both validated for correctness and queried for model properties and components, it is ideal for model sharing and curation, as well as for answering what and how in a machine-readable manner.

To this end Neurounit offers a SciUnit Candidate subclass called NeuroConstructModel. NeuroConstruct [[ref]] is a simulation manager that starts with NeuroML models and then simulates them in any of the conventional simulators listed above. The NeuroConstructModel class is instantiated with the path to any NeuroML model created in or imported into NeuroConstruct. Because the family of all models that can be described by NeuroML is vast, the NeuroConstructModel class makes very limited assumptions about the candidate models that they are runnable, and that at least one compartment generates a membrane potential. NeuroConstructModel is then subclassed to test specific NeuroML models.

The Open Source Brain project (OSB, http://www.opensourcebrain.org) curates many such models. In contrast to previous curation efforts such as ModelsDB [[ref]], OSB projects have been converted from their native format into NeuroML, and run on all major simulators for neurons and networks, such as NEURON, Genesis, PyNN, Moose, and others. Thus, OSB models are effectively guaranteed to be runnable. In addition, the degree of concordance between model output (beginning with the NeuroML description) and reference output (from the original source files) is reported for each model, and is generally near perfect. Thus, OSB is an excellent source of models that, in addition to being open source, are described completely enough to make validation possible. The hippocampal CA1 pyramidal cell is represented in several OSB models, and we demonstrate testing using a CA1PyramidalCellModel class (Figure X), deriving from NeuroConstructModel. All OSB models, and indeed any NeuroML model, can be tested similarly. Working

together with OSB is part of our first collaboration, and our integration efforts can be publicly tracked in our fork of the neuroConstruct code repository (http://github.com/rgerkin/neuroConstruct).

Although they span a range of scales, and regardless of their original implementation, all OSB models are formally described using NeuroML, as are each of the model components and sub-components, such as cells, ion channels, calcium stores, etc. These models are regularly executed on OSB servers to ensure that, as they are updated, their output is consistent with previous versions. Therefore, OSB can confirm that they work, and linked journal articles, and on site wiki, and inspection of the native code or NeuroML itself can establish how they work. However, currently there is no mechanism for establishing how well they work, i.e. how well the models perform at explaining data. SciUnit will fill this gap by providing OSB (and the biology community in general) with a means to assess models using data-driven tests in the NeuroUnit library. Consequently, researchers will be able to see not only what is being modeled and how – as OSB currently facilitates – but also how well model output conforms to measured neurophysiological properties. OSB models will be tested using the SciUnit framework, utilizing NeuroUnit tests of our own design as well as those contributed by the community. A similar process can be used to apply SciUnit to other biology sub-disciplines using corresponding community repositories of models from those disciplines.

4.1.2 Capabilities from NeuroTools

NeuroTools (http://neuralensemble.org) is a collection of tools to support all tasks associated with a neural simulation project and which are not handled by the simulation engine written in Python [[ref]]. Specifically, it permits the extraction and analysis of simulation output, such as membrane potential time series, spike trains, etc. NeuroTools is an open source and actively developed projecs, and represent reliable libraries on which neurophysiology tests can be based.

We use these libraries to implement SciUnit Capabilities in NeuroUnit (Figure X). For example, a NeuroTools AnalogSignal object (e.g. a membrane potential time series) has threshold detection method that returns a NeuroTools SpikeTrain object. A NeuroUnit HasSpikeTrain Capability requires that the method getSpikeTrain be implemented. While one could implement this in any number of ways, a NeuroConstructModel does so by making getSpikeTrain wrap `AnalogSignal.threshold_detection`. This is one of many examples in which NeuroTools objects are exchanged between NeuroConstructModel methods. This greatly simplifies test writing, since many basic properties of model output are obtained trivially by using NeuroTools object methods, and these NeuroTools objects are easily extracted from model output using candidates derived from the NeuroConstructModel base class. This is possible because NeuroConstruct already implements the creation of NeuroTools objects from model output.

4.1.3 Reference Data for Tests from NeuroElectro

Answering how well requires validation testing against data. The NeuroElectro project (http://neuroelectro.org) is an effort to curate all published single cell neurophysiology data [[ref]]. Currently, up to 27 electrophysiological properties are reported for XXXX cell types, spanning XXXX single pieces of published data extracted from article tables. NeuroUnit makes it easy to construct Tests that use reported values on neuroelectro.org as reference data. NeuroUnit enables construction of tests using reference data from either single journal articles, or from ensembles of articles about a particular neuron type. The former is illustrated in Figure X.

Data about specific neuron types stored on neuroelectro.org are accessed using the NeuroElectro API and the statistics of that data (mean, standard error, and sample size are typically reported) serve as the reference against which model output is judged. In most cases the data available on NeuroElectro is not sufficient to

judge all aspects of model; however, this data can nonetheless serve to validate basic features of most neurophysiology models, such as membrane potential, action potential width, after-hyperpolarization amplitude, etc. The data may reflect artifacts of the experimental preparation, such as electrode type or recording temperature, but this metadata is increasingly visible in the NeuroElectro project, and test writers are encouraged to identify the data most appropriate to the conditions under which the model is simulated or is intended to represent. In any case, NeuroElectro represents the only publicly curated source of electrophysiological properties of neurons, and thus represents the primary source of reference data for the validation of basic model features. Co-development of the NeuroElectro API through which these data are obtained represents our second collaboration (`http://bitbucket.org/rgerkin/neuroelectro_org`).

4.1.4 A Complete Pipeline

Although we do not intend for NeuroConstruct, NeuroTools, and NeuroElectro to represent the only sources of models, capabilities, and test data, they provide an immediate point of entry into the neurophysiology community and powerful demonstration of our proposal. In the NeuroUnit repository (http://github.com/rgerkin/neurounit) is a runnable script (examples.py) which demonstrates a complete testing pipeline. It selects the CA1 Pyramidal cell model from OSB, simulates it using NeuroConstruct, and tests the widths of action potentials using a NeuroUnit test class called SpikeWidthTest. The script instantiates SpikeWidthTest using data obtained on-the-fly via the NeuroElectro API. Finally, the script computes the results of the tests and prints them.

4.1.5 Creating New Candidates, Capabilities and Tests

As described above, NeuroUnit provides base classes to enable rapid generation of Candidates, Capabilities, and Tests for neurophysiology data. However these objects can also be created using little or none of the resources described here. All that is required is adherence to the SciUnit interface. For example, a Candidate could implement a run Capability method that executes a MATLAB script and a `get_spikes` Capability method that parses a .csv file on disk, and a Test could use spike rates collected in the lab as reference data. The basic workflow is the same as that shown above for NeuroConstruct models, NeuroTools analysis, and NeuroElectro data.

4.2 Public Dissemination

Concurrent with these efforts, we will create a community portal (SciDash) written in Python for consistency with existing projects and standards in scientific computation. This web application will in turn make use of the SciUnit testing framework, and be discipline-agnostic. The source code for the web application and the framework will be developed openly and released under a permissive open source license continuously. An open release, rather than one implemented solely behind our portal, supports labs or groups that want to test models on their own machines as they are being developed, or to compare competing models on unreleased data. The community portal will be populated with models and tests as they become available, so that a) the progress of the project, including the rate of community adoption, is transparent, and b) there exists a dedicated resource for evaluating the validity of models. Tests written for existing OSB models will naturally be useful for testing models not yet in OSB. We plan to release these tests as part of NeuroUnit, an optional package that makes use of the SciUnit framework. Selections from this suite of tests can then serve as a gold standard by which emerging models are judged, facilitating both the development of those models and subsequent community evaluation.

4.3 Project Milestones

Year 1: We will focus on SciUnit core development, construction of tests from Neuroelectro data, and other data provided by the OSB community. At the end of this period we will have a manuscript in review, describing the idea and preliminary results, and have released a stable SciUnit Python module for interested developers. Year 2: We will begin automating model testing for The Open Source Brain project (OSB), greatly increasing the number and scope of the models tested. We will continue to define SciUnit model capabilities via collaborative development of NeuroTools, and aggregate new datasets to test the capabilities of the OSB models. At the end of this period, OSB will support automated testing and result summaries, and SciDash will contain a wide array of models and tests to browse and visualize. Year 3: We will refine the framework and add extensive documentation to the SciUnit project to encourage adoption. We will actively promote its use at conferences. We will continue to write tests and specify (existing, published) models in NeuroML for execution and testing. At the end of this period we will have another manuscript in review, describing the results of the project and promoting the SciDash portal. At the end of this period there will be sufficient tests available in NeuroUnit to motivate considerable community interest and serve as an example for developers in other disciplines of biology.

4.4 Challenges

The proposal described here faces theoretical and practical challenges to its implementation. Here we describe these and how we anticipate overcoming each one.

4.4.1 Participation from Modeling Communities

Modelers may not want to expose model capabilities for test-taking. We anticipate four solutions: First, interfacing a model to SciUnit requires only writing methods for selected model capabilities. This means identifying the outputs of a model that satisfy a capability, and returning their values. This procedure could be as short as one line of code. It also means identifying the inputs to a model corresponding to configuration parameters. However, the modeler is not required to expose or rewrite any model flow control. Second, we will support multiple environments automatically by using simulator-independent model description languages. In neuroscience, the NeuroML standard represents such a description (Gleeson et al, 2010) – it describes a model in a machine-readable way which allows the same model to be executed any simulator of choice. Models described in NeuroML can be automatically translated for execution on a range of simulators. For those models which are specified using NeuroML, it should be possible to enumerate capabilities of that model, derive a test suite for which it is eligible, and execute those tests in a fully automated fashion. While many neuroscience models are already specified in NeuroML, are far larger number are written directly for simulation in NEURON, the most popular neural simulator [[ref]]. NEURON supports the conversion of NEURON source files directly to NeuroML documents, which opens up thousands of models spanning decades of effort to testings. Efforts are also underway to automate the generation of NeuroML from other popular simulators such as MOOSE (Gleeson, personal communication). Thus for a large and growing number of models, modeler effort is already close to zero. Third, a modeler has a strong incentive to use SciUnit in order to demonstrate in an umambiguous and public way that her model is consistent with data. Fourth, a modeler has a strong incentive to use SciUnit during development (see TDD, above) to guarantee that design choices made along the way do not break the correspondence between the model and the phenomenon it is trying to explain. The existence of a suite of popular tests will represent a gold standard by which a modeler can judge her progress during development.

4.4.2 Participation from Experimental Communities

Experimentalists may not want to write tests derived from their data, or are not comfortable with writing code. We anticipate three solutions: First, rather than demand the use of special formats for data, each test will need only a list of required model capabilities (for selecting eligible models), essential experimental metadata (for configuring models) and a statistical summary of data (for scoring results) be written into each test. By definition each unit test is focused, and does not require the ability to do arbitrary computations on a data set. As an example of the ease of test-writing, suppose that one has evoked by intracellular current injection in a cell 100 action potentials and wishes to write a test concerning the width of these action potentials. Writing the test then consists of selecting ReceivesCurrent and ProducesActionPotentialShape capabilities (one line of code each), typically computing the mean and variance of action potential widths in the data (one line of code), specifying the parameters of the current injection, e.g. the amplitude and the duration (two lines of code), and finally selecting a scoring mechanism, e.g. Must be ¡ 1 standard deviation of the mean value (one line of code). Most of the heavy-lifting is done by the interface. Second, as data-sharing becomes more ubiquitous, this task can be distributed across a large number of scientists, including non-experimentalists interested in data analysis or testing their own models. Third, a strong incentive to write tests for ones data exists: the ability to identify models that explain ones data, giving the data clear context and impact.

4.4.3 Diversity of Levels and Kinds of Models and Data

How can one framework deal with so many kinds of topics in biology? First, we solve this by providing an interface that allows modelers to express the capabilities which their model possesses. The set of all capabilities so described determines the range of tests that can exist, and the set expressed by one model determines the range of tests that this model can take. Hierarchies of scale are embedded in the inheritance of capability classes. For example, an answer to the question Does this model have action potentials requires a yes answer to the question Does this model have neurons. Consequently, the incompatibility of a test-requiring-action-potentials for a model-lacking-neurons is known without explicit tagging. Conversely, a model with a Hodgkin-Huxley capability also inherits a voltage-gated capability, because the former implies the latter. Second, expressing models in NeuroML naturally addresses diversity of levels (i.e. scales) because NeuroML is developed in levels, with a hierarchical organization. Thus, models can be sub- or supersets of other models. Third, testing across levels can also be implemented using the Representional Similarity Analysis (RSA) framework (Kriegskorte et al, 2008), which requires only that a model be capable of responding to a defined set of inputs (e.g. stimuli). The similarity matrix for responses within a model and across inputs defines a unique signature for that model, and can be the intermediate output of a unit test. Model scale becomes irrelevant, because test scores are then based on goodness-of-fit between the similarity matrix for the model and that for a corresponding experiment these matrices can be compared directly no matter the model scale because their size depends only on the number of test inputs, not on the details of the systems being studied.

4.4.4 Appropriateness of Models for Validation

It is not the purpose of all models to reproduce experimental findings. Some models are simply a proof of concept that a dynamical system with configuration X will have properties P. We have no intention of challenging those proofs with model validation. But by taking and passing tests that abstract away most experimental details, even very abstract models can benefit from testing. For example, rather than encoding

experimental stimulus and response values using dimensioned units (e.g. mV or pA), a test author could express a test as simply a mapping between sets of numbers. Some abstract models may have unexpected homology to that mapping, thus highlighting the relevance of such models where it may otherwise have been missed. Alternatively, a model may make very specific predictions about experiments, but require a significant amount of contextual information to do so that the test may not provide. Rather than fail models which require this information on tests which do not provide it, we give them an incomplete, i.e. no record of the test result is generated for such a model.

4.4.5 Arbitrary Scoring Criteria for Tests

A raw test score is computed from goodness-of-fit to data, and a pass/fail mark from that score. At both stages there is room for arbitrary design choices that will benefit some models at the expense of others. First, however, many goodness-of-fit functions have, for most inputs, identically rank-ordered outputs, meaning that there will be few cases where these design choices will cause an otherwise passing model to fail and vice versa. For example, using Z-scores will yield the same rank-ordering as using p-values derived from those Z-scores. Indeed, non-developers can ignore quantitative differences between model scores, with focus given instead to the rank ordering of those scores. Second, since the SciUnit project is open, it is straightforward to clone a test, change the statistical choices used in the scoring mechanism, and use the modified test. With transparency regarding how each test is conducted, the community can then decide which version of the test is most appropriate. This process will be open and documented on the SciDash web portal. This approach is consistent with the aims of the project: the community can decide on what models should do, and the framework can determine whether it does those things.

4.4.6 Reliability of Data Underlying Tests

Unreliable data will lead to tests that even perfect models will not pass. First, as with any system of model evaluation, it is incumbent upon the community to evaluate the experimental design and techniques involved in producing data, and to discount data that are known to have been produced using questionable methods. The SciDash web portal will allow for community moderation, permitting users to rate and comment on tests, indicating those believed to be derived from suspect data. Second, models cannot and should not be expected to fit perfectly to data, when data is merely a random draw of finite sample size from a true distribution. This limitation can be addressed by making explicit the uncertainty in the data, by asking how well one data set can validate its own experimental replications (Kriegskorte et al, 2008). A model cannot be expected to validate any better than a second experiment performed on the same system, so test output may be transformed to reflect that a goodness-of-fit matching that seen across experimental replications represents a perfect score.

4.4.7 Computational Efficiency

Some large models may take a long time to execute, and to subject that model to many tests could be quite computationally intensive. To lighten the burden, we prioritize minimal re-execution of the same model in SciUnits design. This means that a set of tests that require as input the same model output should only require the model to be executed once. The model output, generated by execution of the first test, can be cached and stored for use by similar tests. This can be done easily be storing instances of model execution as Sage worksheets on the SciDash portal. We have also implemented caching in the sciunit.utils module, by storing Candidate instances and associated run data in an optional database backend.

4.4.8 Occam's Razor

The ability of a model to explain the data is typically weighed against the complexity of the model – simpler models being better, ceteris paribus. Model complexity can have many definitions, so the framework will report several complexity metrics for models. These include: the number of lines, instructions, or operations in the model memory use during model execution. average CPU load during model execution. number of model parameters Larger, longer, and more expressive models may be considered more complex (McCabe,1976). There are several ways to represent the tradeoff between model validity and complexity. One can report it in tabular form (e.g. Table 2). A scatter or line plot, with the best models being in the high validity / low complexity corner of the plot, is also informative. The set of models which dominate others, that is, which have both higher validity and lower complexity than their rivals, are represented as a validity vs. complexity front, showing only those models which have the highest validity for each level of complexity, similar to the visualization used in the symbolic regression package Eureqa (Schmidt and Lipson, 2009). Ones judgment is then required to weigh the relative importance of validity vs. complexity for ones application.

4.4.9 Expansion Into Other Areas of Biology

After proving its utility in neurophysiology, we would like SciUnit to expand first across neuroscience and then into other biological sciences. Since the core framework is discipline-agnostic, the only obstacles are community participation and model description. As with neurophysiology, community participation begins with enumerating the capabilities relevant to a sub-discipline, and then writing tests. Model description can expand within NeuroML (which already covers multiple levels and approaches within neuroscience) and NeuroUnit tools can begin to incorporate libraries for neuroimaging (NiBabel [[ref]]), neuroanatomy (NeuroHDF) and others. SEDML (Hucka et al, 2003), a model description language for systems biology, will facilitate expansion outside of neuroscience. This transition will be facilitated by parallel efforts in the NeuroML community to interface with SEDML (Crook, unpublished).

5 Community and Educational Outreach

5.1 K-12 Education

The cornerstone of a basic science education is learning the scientific method. Unfortunately, the process by which the scientific method is applied both within and across labs may be too informal be recognizable to students. The SciDash web portal will provide an example of the scientific method in practice, determining which hypotheses (models) can withstand the scrutiny of evidence (can pass tests). Revision of hypotheses to match evidence will also be transparent, as SciDash will show and optionally group test results for each variant of a model. Indeed, OSB already tracks and makes available revisions to models, so the addition of a testing framework and transparent test results to that project would complete both the implementation and presentation of the scientific method for selected models in neuroscience. The ability to visualize the scientific method at work from any computer in the world will represent a major step forward in science education. To make the educational relevance of a validation testing framework more clear, consider teaching the history of advances in cosmology according to the schematic of Table 2:

In Table 2 we consider 5 models of stellar and planetary motion, and describe 4 validation tests derived from relevant data. Validation testing recapitulates the history of cosmology in a way that shows the scientific method to be an on-going process. The Geocentric model of Claudius Ptolemy (Ptolemy, 150) would

pass a test derived from Babylonian records of solar and planetary motion. Ptolemys model fails all the other tests shown here, including a test constructed from Tycho Brahes more meticulous measurements of planetary motion (Kepler, 1627), which were inconsistent with Ptolemys notion of perfectly circular orbits. The Copernican Heliocentric model (Copernicus, 1543) also predicts circular orbits and thus fails Brahes test, but it is far simpler than the Ptolemaic model, dispensing with notions such as epicycles. Keplers laws of planetary motion both permit and explain these elliptic orbits (Kepler, 1609), as well as Galileos unanticipated discovery of moons taking elliptic orbits around Jupiter (Galileo, 1610). Newtons gravitational model passes these tests and does so by more succinctly by unifying Keplers laws under one principle gravity (Newton, 1687). One of the earliest challenges to Newtons model came in measurements of perihelion precession of Mercury (Le Verrier,1859). It was not until the discovery of General Relativity that this problem was resolved (Einstein, 1916). Interestingly, Einstein proposed 3 tests of his theory, of which one was based on Le Verriers data. While this account may be simplistic compared with the complexities of validation testing in modern biology, it would be appropriate for teaching the essentials of the scientific method to a high school or college student. Although teaching is not part of the budget request for this proposal, future proposals focused on using SciUnit to teach the process of science will be forthcoming.

5.1.1 Journalistic Media

When providing coverage of new scientific theories, non-scientist members of the media have no reliable way to determine the importance or quality of those theories. Even consulting scientists from the field may be unreliable, due to issues of bias or because the consultant lacks the appropriate expertise. In contrast, SciDash would – for any model that it covers – provide an unbiased way for a member of the media to immediately identify the scope of a model and the range of data it explains, and contrast this with previous efforts. With the original sources of each model and test well-documented on SciDash, relevant contributors could be contacted for further comment or explanation. Site-embedded commentary provided by modelers and test-writers will also be a helpful media resource.

We believe that the competitive nature of SciDash will be appealing both to scientists and to the lay community. Competitions for machine learning [[ref]], solar cars [[ref]], and autonomous vehicles [[ref]] already draw considerable media coverage and we believe that the brain is of no less interest to the public. Another feature of public competitions is that they welcome teams who may not yet have academic credentials, such as students, both to learn the craft and possibly to demonstrate key insights that had not previously been recognized in the professional scientific community.

6 Personnel and Coordination

A scientific software framework succeeds in proportion to its rate of adoption, which is driven in part by: a) Relevance to the needs of both experimental and theoretical scientists. b) Quality of architecture and usability. c) Conformance to accepted data and modeling standards in the targeted communities. d) Integration with existing software tools. e) Applicability to outstanding questions in a field. f) Community access (i.e. an open and accessible internet presence).

6.1 Personnel

We have the appropriate team members to meet the six criteria above.

PI: Richard C Gerkin, PhD Title: Assistant Research Professor, School of Life Sciences, Arizona State University (ASU) Expertise: Neurophysiology, computational models, informatics, web development. Role (Time): Will coordinate all project activities (25 hrs/week). Detail: As the main PI, he will coordinate all project activities, and be responsible for all output including code, databases, and manuscripts. Dr. Gerkin will o a) coordinate with the NeuroElectro project and solicit other sources to obtain physiology data, format and annotate this data, and construct tests from the data. Experience as both an experimentalist and a modeler will facilitate this objective. o b) write the NeuroML binding for the testing interface (Conformance) o c) subject models from OSB (and similar models publicly available) to these tests o d) develop and maintain the SciDash website (Community) o e) coordinate with the developers of OSB to implement automated SciUnit testing.

Co-PI: Sharon M Crook, PhD Associate Professor, School of Mathematical and Statistical Sciences, ASU Associate Professor, School of Life Sciences, ASU Expertise: Neuroinformatics, Computational Modeling Role: Interface of SciUnit with NeuroML and Open Source Brain Detail: Access to the broader neuroscience community requires the use of an accepted model description standard. We naturally chose NeuroML due to its advanced state and broad coverage of multiple scales. Dr. Crook is funded by NIH to maintain and support NeuroML, and has committed her support to helping us use NeuroML for model specification (Conformance).

Co-PI: Jonathan Aldrich, PhD Associate Professor, School of Computer Science and Institute for Software Research, CMU Expertise: Software engineering, software verification and validation, human factors Role (Time): Guide overall software architecture (2 hrs/week) Detail: Dr. Aldrich will supervise and train Mr. Omar. His expertise in software design for science and engineering applications will be key in this role, and assure quality. While Mr. Omar will be writing code, Dr. Aldrich will be providing guidance on the overall structure of the implementation. Dr. Aldrich is already Mr. Omars graduate supervisor (funded by other sources), but for this project Dr. Aldrich will take on the additional training responsibilities specific to the focus of this proposal.

Junior Personnel, Cyrus Omar Predoctoral Fellow, School of Computer Science, CMU Expertise: Computational models and computer science Role (Time): Core framework design and development (25 hrs/week) Detail: Mr. Omar is a senior graduate student who has expertise in both computational modeling in neuroscience and software infrastructure for science. He will be responsible for guiding the overall software architecture of the project and developing and maintaining the core SciUnit Python module. He will get feedback from Dr. Gerkin about the outcomes of model testing, which will inform revisions to the code.

Collaborator, R. Angus Silver, PhD Title: Professor of Neuroscience and Wellcome Trust Senior Research Fellow, Faculty of Life Sciences, University College, London Expertise: Neurophysiology, computational neuroscience and neuroinformatics Role: Interface of SciUnit with NeuroML and Open Source Brain Detail: A frequent collaborator of Crook, Silver maintains Open Source Brain (OSB), the largest standards-driven respository for models in neuroscience, for which he is funded by The Burroughs Wellcome Trust. The OSB team has welcomed us to test SciUnit on models described at OSB (Integration). This provides us a wide range of neuroscience models specified in NeuroML (Applicability). This will in turn expose the project to the international neuroscience modeling community and serve as proof of concept for expansion into other areas of biology.

6.2 Means of Coordination

Dr. Gerkin trained in Pittsburgh and maintains ongoing collaborations with Dr. Aldrich, and Mr. Omar. Dr. Gerkin works at ASU and regularly meets with Dr. Crook for Math Biology group meetings there. Because the existing projects that form the basis for each collaboration are both a) well-documented, and b) available under an open license, there should be no barriers to code sharing and collaborative development. The popular and powerful Github platform will be used for code development and communication between developers.

7 Results from Prior NSF Work

Sharon Crook was PI on an NSF grant that

Jonathan Aldrich has been a PI or co-PI on four prior NSF grants. NSF CAREER grant CCF-0546550, "Lightweight Modeling and Enforcement of Architectural Behavior" (2006-2010), focused on the Scholia approach to extracting a run-time architecture from object-oriented code, as well as analyzing architectural behavior. 12 major journal or conference publications – note that major conferences are more prestigious than journals in CS – as well as 12 workshop articles emerged from the grant, including (Abi-Antoun et al, 2007, Abi-Antoun et al, 2008; Abi-Antoun and Aldich, 2008, Abi-Antoun and Aldrich, 2009). NSF grant CCF-0811592, "CPA-SEL: Practical Typestate Verification with Assume-Guarantee Reasoning" (2008-2011), funded fundamental advances in lightweight typestate verification approaches. These were described in 7 major publications, including (Bierhoff and Aldrich, 2007; Beckman et al, 2008; Bierhoff et al, 2009; Jaspan and Aldrich, 2009; Sunshine et al, 2011). NSF grant CCF-1116907, "SHF:Small:Foundations of Permission-Based Object-Oriented Languages" (2011-2014), is only a year old, but has already yielded a paper in the premier publication venue for programming languages describing a novel type system for modularly reasoning about aliasing in object-oriented programs (Naden et al, 2012). These grants resulted in several open-source software tools for research and education, including ArchJava, Plural, Crystal, and SASyLF (see references cited). Aldrich has just been awarded NSF grant "TUES: Collaborative Research: Teaching Software Modularity through Architectural Review," but only a month has passed since that award.