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 models whose scope is of interest and characterizing their validity against a continually growing body of available evidence.

A new model is typically proposed 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, supported by a handful of relevant equations and figures, and judged by peer review. Ideally, reviewers ensure that the model's predictions are consistent with the available and relevant data and compare each model against previously published models. This job is becoming increasingly difficult; although authors may refer to relevant experimental data and provide citations into the literature, these are generally incomplete and biased, in that authors often focus on data favorable to their models and compare them to implausibly simple hypotheses.

If modeling papers contained detailed comparisons to all related experiments and competing models, publications would become encyclopedic and their main points obscured. A strength of model publication today is its 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 model is intractable using a publication alone. Publications tell us **how** a model works but are less effective at comprehensively reporting **which** goals it achieves and **how well** it achieves them. This problem is exacerbated as more data is gathered following publication. Although model validity may change in light of this new data, there is no systematic process in biology today for re-evaluating existing models. Although new data and its most important theoretical implications propagate informally through a scientific community or appear in periodic reviews, the original publications – a resource of first resort for new scientists and onlookers – are cited "as-is" in perpetuity.

We can distill the central problem discussed here as this: the process of scientific model validation today is not sufficiently rigorous, comprehensive, or ongoing. This hampers scientists' ability to make accurate predictions about experiments, compare models, and precisely identify outstanding research problems. To overcome these obstacles, we propose systematizing the model validation process by creating software and associated cyberinfrastructure dedicated to scientific model validation, supplementing today's system of peer-reviewed scientific publications to better support the modern shift toward science at a large scale.

1.1 Existing Efforts

There are several well-developed facilities for data and model sharing in biology, but few if any facilitate evaluation of models against data directly. For example, the Collaborative Research In Computational Neuroscience (CRCNS) data-sharing website [1], and the Open Source Brain (OSB) repository [2] are facilities for data and model sharing in neuroscience, respectively. The CRCNS website hosts several excellent data sets of relevance to computational models; however, the data is not organized or annotated to indicate which models may be capable of predicting that data, nor how well they do. The OSB is a resource for standardized model descriptions that permit reliable execution. 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. We aim to bridge resources like these, increasing the usefulness of 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. Kaggle [3] drives model development by organizing competitions where training data is made public and competitors submit competing algorithms, compared automatically by cross-validated accuracy on an unrevealed test set. The success of Kaggle [4] shows that open competitions are effective, and that this paradigm of "modeling as a competition" attracts data scientists across traditional discipline boundaries. However, models developed in this way fit only a few general cases in machine learning: classification, regression, and clustering. The validation criteria are straightforward in these domains. Discipline-specific competitions having a similar structure but with more sophisticated validation criteria within biology have also resulted in important advances. For example,

the quantitative single neuron modeling competition [5] has helped us understand the complexity-accuracy tradeoff among reduced models of excitable membranes; the "Hopfield" challenge [6] famously illustrated the challenges of identificationfacing computational neuroscience; the Diadem challenge is advancing the art of neurite reconstruction [7]; and examples from other subfields of biology abound [8]. Our challenge is to develop a general framework in support of distributed data-driven model validation workflows, in the flavor of these kinds of competitions, but where the validation criteria themselves are also specified collaboratively. Initially, we will focus on validation challenges in the biological sciences, particularly neuroscience.

2 Outcomes and Products

Our framework is organized around simple executable *validation tests* that compute agreement between a prediction generated by a computational model and an experimental observation. The scope of a model is identified with the set of tests that a model is capable of taking and validity is identified with the set of these tests that the model successfully passes. This methodology is inspired by the ubiquitous practice of *unit testing* in software engineering. A unit test evaluates whether a portion of a computer program (often a single function) meets a simple correctness criterion. A suite of such tests *covering* desired program behaviors validates overall program functionality. Due to the narrow scope of each test, failed tests help precisely identify which parts of a program are not functioning correctly. Developers often write unit tests before writing the program itself, following a methodology called test-driven development (TDD) [9]. By writing them early, tests serve as a partial program specification, guiding its development. This allows developers to measure progress simply by looking at the proportion of tests that pass at any point during development. When modifications are made, developers can be confident that *regressions* have not appeared by ensuring that that all tests that passed before a change continue to pass after it. The success of unit testing and TDD in practice suggests that validation testing may be a *practical* approach to systematic model validation.

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 must 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 (called a *spike count*) and the distribution observed in the data, in response to the same stimuli. This test would ask any candidate model being tested to produce a predicted spike count for each stimulus and check whether the prediction fell within the empirically-observed distribution of action potential counts observed 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 have used to (in)validate their model.

2.2 Validation Testing with *SciUnit*

The first product of this proposal is a simple validation testing framework to support examples like the one above. This framework, called *SciUnit*, is written in Python and is designed to make test construction (from data) and execution as easy and flexible as possible. Testing in SciUnit proceeds in three phases (Figure 1):

1. 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.
2. 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.
3. 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 model's 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 test's `required_capabilities` attribute. (2) Test Execution: The test's `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.

2.3   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.

2.4   Community Workflow

Unit tests are produced by the community of developers involved in the development of a program. Correspondingly, members of a scientific community can collaboratively produce suites of validation tests in common source code repositories. Each test specifies the model requirements, and provides 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). Model performance on a collaboratively curated suite of validation tests can serve as justification of claims regarding the model's validity as a description of the scientific system characterized by the test suite. This workfLlow continuously produces a summary of the field indicating which models are capable of taking each test, i.e. whether the test falls within the model's

scope, and for each capable model, how well it performed, i.e. its validity. Model usefulness becomes a function of the weight given to each test, determined per investigator or by community consensus. This test-driven scientific workflow leverages the desire of modelers to promote their models' virtues – test suites represent clear *challenges*, issued by experimentalists to the community, and passed tests certify success for the modelers. As more data is gathered and test suites are refined, past models can be tested continuously againt current empirical knowledge, in a fully-automated manner because the interface between tests and models is fixed.

2.5   *SciUnit*: A Simple Validation Testing Framework

Conceptual and practical simplicity – the absence of "heavy-lifting" – is an essential requirement for community participation. Our design does not require the release of raw data or storage in standard formats. Instead, salient aspects of the data are abstracted away behind a test definition. Similarly, 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 model capabilities. Each community can collaboratively specify model interfaces capturing high-level operations of their preferred modeling formalisms. Existing modeling standards (e.g. NeuroML [10, 11]) and interface definition languages (IDLs) [?]are leveraged to simplify this process. **Thus, the first product of this proposal is an object-oriented validation testing framework called *SciUnit*, written in Python, that makes test construction (from data) and test-driven model execution easy and flexible.**
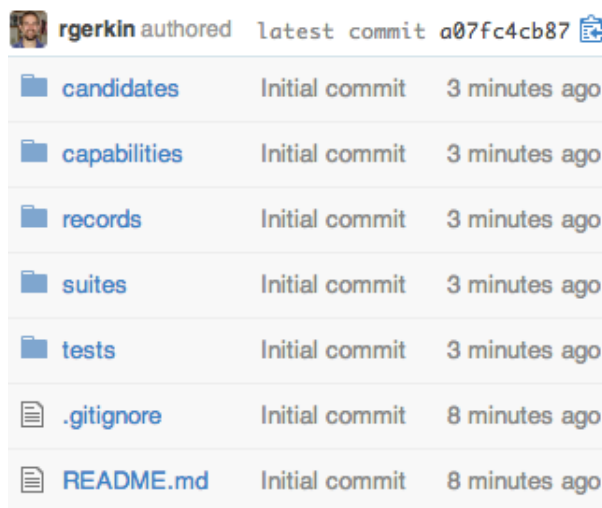
2.6   *SciDash*: A Community web application

This framework is most effective when the current state of a research area is represented by a collection of tests and models. This requires coordination among research groups, so community-oriented cyberinfrastructure to support test suite creation and summarization, building upon *SciUnit*, is essentiall. We propose a service, called *SciDash*, utilizing existing infrastructure for coordinated development of software repositories, focusing on GitHub [12] [13]. A suite of related tests will be contained in a GitHub repository with a stereotyped high-level structure that allows the *SciDash* web application to discover, index and summarize it on an ongoing basis (Fig. 2.5). The portal, a web application written using the pythonic Django framework [14] will serve as

Figure 1: A *SciDash* repository on GitHub

a central location where scientists can discover relevant test suites, determine test requirements, and summarize the results of test execution on submitted models. Test results can be visualized as a "record matrix" composed of large numbers of model/test combinations (Table 2 provides a toy example). Each row in this matrix will contain results for all tests taken by one model and would serve as clear evidence of that model's scope and validity. Models, tests, and records will be stored on GitHub, pointed to by hyperlinks in the record matrix. *SciDash* will serve as a public record of competition among models, facilitating and encouraging data-driven development among modelers. **Thus, the second product of this proposal is a web application called *SciDash*, whose back-end can index test suite repositories collaboratively developed on GitHub, and whose front-end offers users a simple environment for discovery and evaluation of models, tests, and results.**

2.7   *NeuroUnit*: A Suite of Tests for Neurophysiology

To demonstrate the utility of these tools, an initial case study must be conducted. Because we have substantial experimental and computational neurophysiology training and expertise, we target the work toward

4

this domain. By using machine-readable models from the Open Source Brain repository, and corresponding machine-readable data from resources like The NeuroElectro Project, our efforts will produce a body of useful tests that also serve as a demonstration of the test-driven workflow that other areas in the biological sciences can leverage. **The third product of this proposal is *NeuroUnit*, a large suite of data-driven tests and model interfaces for a representative set of canonical neurophysiology models, all publicly available. A case study examining the implementation of *NeuroUnit* will inform development of domain specific test-suites in other biology sub-fields.**

3 Preliminary Activities

We have developed most of the core testing framework, *SciUnit*, under a free, open-source license at the project development repository [15].

3.1 *SciUnit* Implementation

We implemented *SciUnit* using the Python programming language [**?**] due to its widespread adoption across the quantitative sciences. Python also supports interoperability with other major languages used within science, including C, R [16], and MATLAB [17]. The functionality described below could also be readily translated into any programming language with comparable facilities.

3.2 Validation Tests

Testingin *SciUnit* proceeds in three phases (Figure 2):

1. 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.
2. Test Execution: The model is executed to produce output by interaction of the test with model capabilities. If the model is described according to a standard specification, e.g. NeuroML, this corresponds to loading it into a supported simulator, executing it, and reading simulator output.
3. Output Validation: 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 `sciunit.Score` (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`. One *test* is an instance of that subclass, with the needed parameters as constructor arguments.

In Figure 3, the `RateTest` class implements a test family that compares the output "firing rate" (action potentials per second) of a neuronal model, driven by a given somatic input current, to an empirical mean and standard deviation (s.d.) of the firing rate derived from a set of repli-
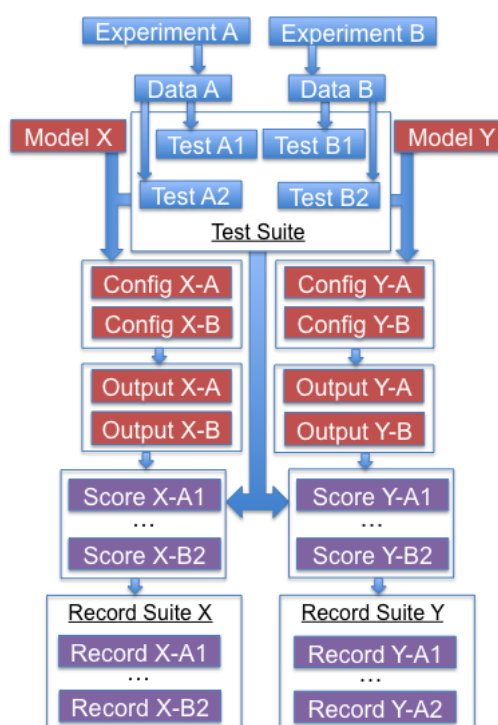

5

Figure 2: Workflow in *SciUnit*. i) Each experiment generates data, ii) From each set of data, many validation tests can be generated from test families (not shown); an ensem-

cated experiments. The test produces a boolean score indicating possible agreement within one s.d. 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.) This parameterization creates a test from the `RateTest` test family. On lines 6-9, 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 s.d on line 17 and a boolean score is produced on lines 18-22. 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 model's output firing rate and the provided mean and s.d. alongside the result, by specifying the `related_data` parameter.

Check line numbers above.

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

```
CA1_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(CA1_cell_rate_test, CA1_cell_model)
```

The `sciunit.run` function operates in three phases, implementing the abstractions from section 3.2:
1. Capability Checking: The model is verified as capable of taking the test by checking each capability in the test's `required_capabilities` attribute.
2. Test Execution: The test's `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. Here the goodness-of-fit is all-or-none, i.e. pass/fail.

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:

```
spike_tests = sciunit.TestSuite([SpikeWidthTest, SpikeHeightTest])
```

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 as in Table 2.

3.4   Candidates and Capabilities

A *candidate* is the entity to be tested. In practice, this will usually implementat a particular scientific model, so *candidate* and *model* can be used interchangeably; we distinguish them to allow for the possibility that a candidate itself be a dataset subject to testing. For clarity, we will refer to candidates as models herein. A model is represented by an instance of a class inheriting from `sciunit.Candidate`. In many cases, models can be grouped into parameterized *model families*, as with tests, with parameters passed to an analogous 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 validity. A *capability* is a contract guaranteeing that a model is able to produce results of a particular form. We model capabilities as Python classes containing unimplemented methods. In other

Figure 3: A single neuron firing rate test implemented in *SciUnit*

```python
class RateTest(sciunit.Test):
    def __init__(self, mean, std, input_current):
        self.mean, self.std, self.input_current = \
        mean, std, input_current

    required_capabilities = (
        neurounit.Capabilities.ReceivesCurrent,
        neurounit.Capabilities.ProducesFiringRate
    )

    def run_test(self, model):
        model.set_current(self.input_current)
        # Implementation guaranteed by capability check.
        rate = model.get_firing_rate()

        mean, std = self.mean, self.std
        result = (mean - std) <= rate <= (mean - std)
        return sciunit.scores.BooleanScore(result, related_data={
            "rate": rate,
            "mean": mean,
            "std": std
        })
```

object-oriented languages, this corresponds to an *interface* or an abstract class.

Let us consider the capabilities of a model publicly available on the Open Source Brain website [18]. In the system being modeled, recorded data consists of the 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". Corresponding model capabilities might include `ReceivesCurrent`, `ProducesFiringRate`, and `TimeIntegrable`, the last of these indicating that the model can be integrated over time.

Many capabilities must be enumerated to span a community's modeling formalisms. In the simple example we present here, these three capabilities suffice. As two of them are domain-specific (to neuroscience), we place them not in *SciUnit* itself but in *NeuroUnit* in the module `neurounit.capabilities`. In contrast, `sciunit.capabilities` contains only domain-independent model capabilities like `TimeIntegrable`.

A model is represented by an instance of a class inheriting from `sciunit.Candidate`. In Figure 4, the CA1 cell model family implements the above capabilities, which both permits testing and importantly determines which tests are within its scope. A models implemented at a different level of detail may have different capabilities and thus be eligible to take a different set of tests.

In Figure 4, the model produces a firing rate after being provided a current. The capabilities a model must implement are determined by its base classes; it multiply inherits some `sciunit.Candidate` subclass (here NeuroConstruct.Model, described in 4.1.1), and some number of `sciunit.Capability` subclasses. The model then expresses methods (such as `set_current`) that override (and implement) the abstract, unimplemented methods provided in the corresponding `sciunit.Capability` subclasses. Any capability the model inherits must have all if its associated methods implemented. Failure to do so will result in a `NotImplementedError`, indicating a model scope less than claimed by its inheritance.

A *model family* is initialized with parameters (`i_leak` in the case of the model in Fig. 4) to produce a *model*. A test does not specify these parameters, however a tester may nonetheless subject a range of models from a model family to the same test. This is done by repeatedly instantiating the model with different parameters (for example, in a loop). If a particular parameter value in the model is required for a test to be sensible, this can be expressed in the list of `required_capabilities`. For example, if the test is only passable

Figure 4: A candidate class corresponding to a CA1 Pyramidal Cell model from Open Source Brain

```
1   class CA1PyramidalCellModel(NeuroConstructModel,
2                               capabilities.ReceivesCurrent):
3       """CA1 Pyramidal Cell model from /neuroConstruct/osb/hippocampus/
4       CA1_pyramidal_neuron/CA1PyramidalCell"""
5       def __init__(self,**kwargs):
6           # Put any other initialization here.
7           project_path = os.path.join(candidates.putils.OSB_MODELS,
8                                       "hippocampus",
9                                       "CA1_pyramidal_neuron",
10                                      "CA1PyramidalCell",
11                                      "neuroConstruct")
12          candidates.NeuroConstructModel.__init__(self,project_path,**kwargs)
13          super(CA1PyramidalCellModel,self).__init__(**kwargs)
```

by models corresponding to a particular pharmacological milieu (e.g. with some fraction of ion channels of a certain type blocked), then `required_capabilities` might include `ModifiesConductance`, exposing a method `set_conductance`. The test specification could also include run-time arguments for models, which could correspond to a waveform of injected somatic current, or alternatively to training data for models implementing completing algorithms. Figure 2 illustrates the testing workflow, and Table 1 provides definitions of terms.

Table 1: Glossary of Terms

Data	The measurements yielded by an experiment.
Model	An executable function that aims to explain, reproduce, or predict experimental data. More generally, a candidate.
Model Family	A class that generates a set of related model instances. Each model instance from one family differs according to the set of parameters (corresponding to a particular experimental design, or set of training data) with which it was initialized.
Validation	The process of assessing the agreement between model output and experimental data.
(Validation) Test	A function that executes a model and scores the model's output against a summary of experimental data. One experiment can inform the development of many tests.
Test Family	A class that generates a set of related test instances. Test instances from within one family differ only quantitatively, according to the data used to initialize them. All tests from one family attempt to validate the same property of a model.
Test Suite	A collection of tests, each of which attempts to validate a different property of a model.
Score	A value summarizing the performance of a model on a test. This can be a measure of the goodness of fit between model output and experimental data.
Scope	A list of tests that a model is eligible to take, indicating the range of experimental data that the model might be able to explain or replicate.
Capability	A contract that states that a model is able to produce results of a particular form. For example, a Hodgkin-Huxley neuron model is capable of producing a membrane potential time-series, whereas a model without a notion of voltage is not.
Record	An indication of the test score achieved by a model, along with any initialization conditions.
Record Suite	A list of records associated with a model. This summarizes the scope and validity of the model.

3.5 *SciDash* Development

We have also begun work on the *SciDash* web application, which point to models and tests and aggregate test results for scientific communities. A development version is currently live [19]. It is populated with models and tests from the educational illustration in Table 2. This application has several key features. **First**, it provides a "dashboard" view of the state of a field by displaying a matrix of records for selected model/test combinations, i.e. those stored in a given GitHub repository (Fig. 2.5. **Second**, each record will link to the tests results (stored in the repository), displaying line-by-line the executed code and interme-

diate output statements, as well as the underlying models and tests. One possibility is the use of the Sage notebook [20], whose "worksheets" facilitate visualization and storage of executed code. 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. We support open authentication via existing web applications (Google, Yahoo, Twitter, etc.), lowering the barrier to participation. The *SciDash* backend will consist of a MySQL relational database, updated regularly using the GitHub API to populate record matrices.

Is this the appropriate level of detail?

4   Research and Development Plan

In order to 1) accelerate development and testing of *SciUnit* and 2) populate *SciDash* with models and tests, we will focus during the funded period on development of *NeuroUnit*, to serve one discipline (neurophysiology), without compromising generalizability. This will facilitate rapid feedback from an experimental community we know well as we iterate through the software development cycle.

4.1   Leverage of existing resources for the development of *NeuroUnit*

Here we describe standard neuroinformatics tools we have adopted to develop *NeuroUnit* [21].

4.1.1   Candidates from NeuroML Models

NeuroML is a standardized model description language for neuroscience [11]. It permits many neurophysiological/neuroanatomical models to be described in a simulator-independent fashion, and these models can be run across many popular simulators due to emerging inter-conversion capabilities of the NeuroML API. Because NeuroML is an XML specification, model descriptions can be validated for correctness and queried for model properties and components, exposing potential capabilities. It is ideal for model sharing and curation, as well as for answering both *what* and *how* programatically.

NeuroConstruct [22, 23] is a simulation manager that starts with NeuroML models and does what is required to hand off simulation to any of several neural simulators. To this end, *NeuroUnit* offers a `sciunit.Candidate` subclass called `NeuroConstructModel`. `NeuroConstructModel` is instantiated with the path to any NeuroML model created or previously imported into NeuroConstruct. Because the set of models that can be described by NeuroML is vast, the `NeuroConstructModel` class makes limited assumptions about candidate models: that they each is `TimeIntegrable`, and `HasMembranePotential`. It is then subclassed to test specific NeuroML models. The Open Source Brain project (OSB, [2]) curates many such models described in NeuroML. In contrast to previous curation efforts such as ModelDB [24, 25]), OSB projects are converted from their native format into NeuroML, and run on major neural simulators such as NEURON, Genesis, PyNN, Moose, and others. The degree of concordance between model output (beginning with the NeuroML description) and reference output (from native simulator source files) is reported for each model. Thus, OSB is an excellent source of models that, in addition to being open source, are described completely enough to enable validation. The hippocampal CA1 pyramidal cell is commonly modeled, and we demonstrate testing using a `CA1PyramidalCellModel` class (Figure 4), deriving from `NeuroConstructModel` and implementing a CA1 cell model on OSB [18]. Importantly, this implementation consists of little more than "wrapping" the functionality of the existing model; The code shown in Fig. 4 is all that is required for a basic implementation. All OSB models, and indeed any NeuroML model via NeuroConstruct, 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 [26].

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 *do* work, while linked journal articles, on-site wiki, and inspection of native code or NeuroML source can establish *how* they work. However, currently there is no mechanism for establishing *how well*

they work, i.e. how well the models accord with data. *SciUnit* fills this gap by providing OSB (and the larger biology community) with a means to assess models using data-driven tests in the *NeuroUnit* library. A similar process can be used to apply *SciUnit* to other biology sub-disciplines using *NeuroUnit* analogues developed for those disciplines.

4.1.2 Capabilities from NeuroTools

NeuroTools [27] is a Python library supporting tasks associated with analysis of neural data (or model output). It implements the extraction and analysis of simulation output, such as membrane potential time series, spike trains, etc. NeuroTools is an open source and actively developed project, containing reliable algoithms on which neurophysiology tests can be based.

We use NeuroTools to implement *SciUnit* capabilities in *NeuroUnit* (Figure **??**). For example, a NeuroTools `AnalogSignal` object (e.g. a membrane potential time series) has a threshold detection method that returns a NeuroTools `SpikeTrain` object. A *NeuroUnit* `HasSpikeTrain` Capability requires that a method getSpikeTrain be implemented. 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 candidate models derived from the `NeuroConstructModel` base class.

4.1.3 Reference Data for Tests from NeuroElectro

Answering *how well* requires validation testing against data. The NeuroElectro project [28] is an effort to curate all published single cell neurophysiology data [29]. Currently, up to 27 electrophysiological properties are reported for 93 cell types, spanning over 2000 single pieces of published data extracted from journal article tables. *NeuroUnit* makes it easy to construct tests using values reported by NeuroElectro as reference data. Tests can be based upon data from single journal articles, or from ensembles of articles with a common theme (e.g. about a particular neuron type). The former is illustrated in Figure **??**.

Data about specific neuron types can be accessed using the NeuroElectro API and the statistics of that data (mean, standard error, and sample size are typically reported) enable judgement of model output. While NeuroElectro alone is not sufficient to judge all model aspects, it can nonetheless serve to validate basic features of most neurophysiology models, such as resting membrane potential, action potential width, after-hyperpolarization amplitude, etc. While the data may reflect artifacts of experimental preparation, such as electrode type or recording temperature, this metadata is increasingly visible on NeuroElectro, and test writers may identify the subset of data appropriate to the system being modeled. As NeuroElectro is the only publicly curated source of such data, it represents a key component for *NeuroUnit* test constuction. Continued development of the NeuroElectro API, through which data are systematically exposed to test authors, represents our **second collaboration** [30].

Should there actually be an example of workflow with NeuroElectro here?

4.1.4 A Complete Pipeline

Although NeuroConstruct, NeuroTools, and NeuroElectro do not represent the only possible 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 [21] is a runnable script (*examples.py*) which demonstrates a complete testing pipeline. It selects a CA1 Pyramidal cell model from OSB, simulates it using NeuroConstruct, and tests the widths of the resulting action potentials, extracted and computed using NeuroTools, against NeuroElectro data obtained on-the-fly via API, using a *NeuroUnit* test class called `SpikeWidthTest`. The script instantiates `SpikeWidthTest` using the NeuroElectro data, tests the model, and finally computes and prints a test score.

4.1.5 Creating New Candidates, Capabilities and Tests

NeuroUnit provides base classes to enable rapid generation of candidates, capabilities, and tests for neurophysiology data. However these objects can also be created using few or none of the resources described here. All that is required is adherence to the *SciUnit* interface. For example, a Candidate could implement

an `integrate` capability method by wrapping execution of a MATLAB script and a `get_spikes` capability method by parsing a .csv file on disk; a test could be initalized using empirical spike rates collected in the lab. From there, the basic workflow is the same as above.

4.2 Public Dissemination

Concurrent with these efforts, we are creating a community portal (*SciDash*) written using Django, a Python web framework. 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. So long as a GitHub repository has been forked from the main SciDash repository [**?**], and is publicly visible (the default in GitHub), it will be automatically indexed by and visible on SciDash. The community portal will thus point to models and tests as they become available, and display corresponding test results. Thus, there will exist a dedicated resource for evaluating the validity of models, and the progress of the SciUnit initiative, including the rate of community adoption, will be transparent.

4.3 Project Milestones

Year 1: We will continue *SciUnit* core development, and implement *NeuroUnit* tests using NeuroElectro data and capabilities using NeuroTools functions. 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 general use.

Year 2: We will automating model testing for OSB, greatly increasing the number and scope of models tested. We will continue to define *SciUnit* model capabilities via collaborative development of NeuroTools, and aggregate new community-provided datasets to test the capabilities of OSB models. At the end of this period, OSB will support automated testing and result summaries, and *SciDash* will contain a broad array of models and tests to browse and visualize.

Year 3: We will extensively documentation *SciUnit* and associated tools to encourage adoption. We will actively promote its use at conferences and workshops. 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 research program and promoting the *SciDash* portal. *NeuroUnit* will contain a sufficient array of usable tests to motivate considerable community interest and serve as an example for developers in other biology disciplines.

4.4 Challenges

Here we describe theoretical and practical challenges to implementation, and how they can be overcome.

4.4.1 Participation from Modeling Communities

Modelers may not want to expose model capabilities, a requirement of test-taking. We anticipate four solutions: **First**, interfacing a model to *SciUnit* requires only implementing selected model capabilities. This means identifying model outputs that satisfy a capability, and returning their values. It also means identifying model inputs used for parameterization, or run-time arguments, and providing a means to set them. However, each of these procedures may require only one line of code. Importantly, the modeler is not required to expose or rewrite any model flow control. **Second**, we will support multiple environments automatically by using NeuroML [11], a simulator-independent model description language. As we have shown, models described in NeuroML will be easy to test using *NeuroUnit*. Efforts are also underway to automate generation of NeuroML descriptions from the source code used by models written for popular simulators (Gleeson, personal communication). For the NEURON simulator, this process is largely complete. Thus for a large and growing number of models, modeler effort is already close to zero. **Third**, modelers have a strong incentive to use *SciUnit* to demonstrate in an umambiguous and public way that their models are consistent with data. Participation in public modeling competitions (see section 1.1) illustrates this incentive. Fourth, modelers have a strong incentive to use *SciUnit* during development (see TDD, above) to guarantee that on-going development does not break the correspondence between the model and the system it explains. A

suite of popular tests will represent a "gold standard" by which modelers can judge their 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 (as run-time arguments to 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. 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 from `sciunit.scores`, e.g. (colloquially) "Must be < 1 standard deviation of the mean value" (one line of code). Most of the heavy-lifting is done by the interface. An example can be found in `neurounit.tests.SpikeWidthTest`. **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 large number of tests can be automatically generated using the NeuroElectro API, and the continued emergence of systematically organized data repositories will expand these possibilities. **Fourth**, a strong incentive to write tests for one's data exists: the ability to identify models that explain one's 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 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 be executed, and the set implemented by one model determines the range of tests that the 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 `HodgkinHuxley` capability also inherits a `VoltageGated` 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. Analogous hierarchies exist in SED-ML [31, 32], a general systems biology markup language. **Third**, testing across levels could also be implemented using the "Representational Similarity Analysis" (RSA) framework [33], requiring only that a model be capable of responding to a defined set of inputs (e.g. stimuli). A "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 test. Model scale becomes irrelevant, because test scores are based on goodness-of-fit between similarity matrices for model and 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 system detail.

4.4.4 Appropriateness of Models for Validation

Not all models attempt to reproduce experimental findings. Some models serve as proof of concept that a dynamical system with have certain properties. We do not intend to challenge these proofs; however, tests that abstract away most experimental details can inform and illuminate such models. For example, rather than encoding specific experimental stimulus and response values, a test could simply evaluate 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 specific experimental predictions, but require significant contextual information not provided by a test. Rather than fail such models, we give them an "incomplete", i.e. no record of the test result is generated

for such a model, in accordance with the model's scope.

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**, many goodness-of-fit functions have, for a given input set, identically rank-ordered outputs, meaning that such design choices will rarely cause an otherwise passing model to fail and vice versa. For example, the function mapping Z-scores to p-values in monotonic. Indeed, one may ignore quantitative differences between model scores, with focus given instead to the rank ordering of those scores. **Second**, since test repositories are open (e.g. Fig. 2.5), it is straightforward to clone a test and change the statistical choices used for scoring. With each test transparent, a community can decide which test version is most appropriate. This process will be open and documented on *SciDash* via GitHub. Thus, the community decides on what models should do, and the framework then 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**, it is incumbent upon the community to evaluate experimental design and techniques involved in producing data, and to discount data produced using questionable methods. *SciDash* will support community moderation, permitting users to rate and comment on tests, indicating their concerns. **Second**, models cannot fit perfectly to data when that data is a random draw of finite sample size from a "true" distribution. This can be addressed by making uncertainty in the data explicit, or by asking how well a data set can validate its own experimental replications [33]. Since a model cannot validate better (in expectation) than an identical experiment, test scores may be modified to reflect that goodness-of-fit matching that of experimental replication represents a "perfect" score.

4.4.7 Computational Efficiency

Large models may execute slowly, and subjecting such models to many tests could be computationally intensive. To lighten the burden, we prioritize minimal re-execution of the same model in *SciUnit*'s design. Sets of tests requiring as input the same model output only require the model to be executed once. Model output, generated by execution of the first test, can be cached for use by similar tests. For example, model execution instances could be stored as Sage worksheets in the corresponding repositories. We have also implemented caching in the `sciunit.utils` module, by storing candidate instances and associated execution data in an optional database backend.

4.4.8 Occam's Razor

A model's ability to explain data is typically weighed against its complexity – simpler models being better, *ceteris paribus*. Model complexity has many definitions, so *SciUnit* will report several complexity metrics, including: 1) the number of lines, instructions, or operations in the model; 2) memory use during model execution; 3) average CPU load during model execution; 4) number of model parameters. Larger, longer, and more expressive models may be considered more *complex* ([34]). The tradeoff between model validity and complexity can be reported in tabular form (e.g. Table 2); alternatively, a scatter plot, with the "best" models being in the high validity / low complexity corner of the plot, may be informative. The set of models which *dominate* all others, i.e. that have higher validity and lower complexity than their rivals, can be represented as a "validity vs. complexity" front, showing those models with the highest validity for a given level of complexity, a visualization familiar from the symbolic regression package Eureqa [35]. One then weighs the relative importance of validity vs. complexity for one's application.

4.4.9 Expansion Into Other Areas of Biology

After proving its utility in neurophysiology, we would like *SciUnit* to expand 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 [36]), neuroanatomy (Neuro-HDF, [37]) and other sub-disciplines. SED-ML [32, 31] will facilitate expansion outside of neuroscience. This transition will be facilitated by parallel efforts in the NeuroML community to interface with SED-ML (Crook, unpublished).

5 Community and Educational Outreach

5.1 K-12 Education

The cornerstone of basic science education is learning the scientific method. However, the process by which the scientific method is applied in practice may be too informal to be recognizable to students. *SciDash* will provide a visible 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 model variants. Indeed, OSB already tracks and exposes model revisions, so addition of a testing framework to that project would both implement and present 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: here we consider 5 astronomical models and 4 validation tests derived from relevant data. This record matrix, showing implied test scores, recapitulates the history of cosmology, showing the scientific method to be an on-going process. The Geocentric model of Claudius Ptolemy ([38]) passes a test derived from Babylonian records of planetary motion. Ptolemy's model fails all other tests shown here, including one constructed from Tycho Brahe's more meticulous measurements of planetary motion ([39]), being inconsistent with Ptolemy's notion of perfectly circular orbits. The Copernican Heliocentric model ([40]) also predicts circular orbits and thus fails Brahe's test, but is far simpler than the Ptolemaic model, dispensing with notions such as "epicycles". Kepler's laws of planetary motion permit and explain these elliptic orbits ([41]), as well as Galileo's unanticipated discovery of moons taking elliptic orbits around Jupiter ([42]). Newton's gravitational model passes these tests and does so by more succinctly by unifying Kepler's laws under one principle: gravity ([43]). Newton's model was challenged by the observation of perihelion precession of Mercury ([44]). General Relativity resolved this problem ([45]). Interestingly, Einstein explicitly proposed 3 tests of his theory, of based on Le Verrier's precession data. While this account may be simplistic compared with testing practice in modern biology, it would be appropriate for teaching 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 reporting on new scientific theories, lay members of the media have no reliable way to determine the importance or quality of such theories. Direct consultation of scientists may be unreliable, due to bias or lack of focused

Table 2: A record matrix illustrating models and data-driven tests from the history of cosmology. Each row is a model; each column is a test.

	Complexity	**Babylon**	**Brahe**	**Galileo**	**Le Verrier**
Ptolemy	Medium	Pass	Fail	Fail	Fail
Copernicus	Low	Pass	Fail	Fail	Fail
Kepler	Medium	Pass	Pass	Pass	Fail
Newton	Low	Pass	Pass	Pass	Fail
Einstein	High	Pass	Pass	Pass	Pass

expertise. However, *SciDash* would – for any model that it covers – provide an unbiased way for media members to identify the scope and validity of a model, and contrast this with previous efforts. With the original sources of models and tests well-documented, contributors could be contacted for comment or explanation. Site-embedded commentary provided by modelers and test-writers should also be a helpful media resource.

The competitive nature of *SciDash* will be appealing both to scientists and to the lay community. Competitions for machine learning, solar cars, and autonomous vehicles already draw considerable media coverage

Add more discussion of teaching, workshops, etc. to get people to understand and use *SciUnit*, as suggested by the reviewer.

and we believe that the brain is of no less interest to the public. Public competitions also welcome teams who may not yet have academic credentials, such as students, both to learn the craft and possibly to demonstrate key insights not previously 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: **Relevance** to the needs of both experimental and theoretical scientists; **Quality** of architecture and usability; **Conformance** to accepted data and modeling standards in the targeted communities; **Integration** with existing software tools; **Applicability** to outstanding questions in a field; **Community** access (i.e. an open and accessible internet presence).

> Make sure to get each of these terms in the personnel descriptions below as often as possible without being annoying.

6.1   Personnel

We have the appropriate team members to meet the 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. He will also: 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 (**Relevance**)); write NeuroML-related bindings (e.g. `NeuroConstructModel`) (**Conformance**); subject models from OSB (and similar models publicly available) to these tests; develop and maintain the *SciDash* website (**Community**); coordinate with the developers of OSB to implement automated *SciUnit* testing.

[Co-PI: Sharon M Crook, PhD]
Title: Associate Professor, School of Mathematical and Statistical Sciences, School of Life Sciences, ASU
Expertise: Neuroinformatics, Computational Modeling
Role (Time): Interface of *SciUnit* with NeuroML and Open Source Brain (2 hrs/week)
Detail: Access to the broader neuroscience community requires the use of an accepted model description standard. We 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 will help us use NeuroML for model specification (**Conformance**).

[Co-PI: Jonathan Aldrich, PhD]
Title: 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. Omar's 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]
Title: 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 (**Quality**) and developing and maintaining the core *SciUnit* Python module. He will get feedback from Dr. Gerkin about practicalities 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 Wellcome Trust. The OSB team has welcomed us to integrate *SciUnit* into OSB for automated model testing(**Integration**). This exposes a wide range of neuroscience models specified in NeuroML (**Applicability**), and in turn exposes the project to the international neuroscience modeling community, serving as proof of concept for expansion into other areas of biology.

[Collaborator, Shreejoy Tripathy, PhD]

Title: Research Fellow, Dept. of Biological Sciences, CMU

Expertise: Neuroinformatics, computational neuroscience, data mining.

Role: Interface of *SciUnit* with NeuroElectro

Detail: A frequent collaborator of Gerkin, Tripathy maintains The NeuroElectro Project, the largest data repository concerning electrophysiological properties of neurons. This provides us a wide range of data for testing neuroscience models. Tripathy will facilitiate **integration** with NeuroElectro through further development of the API.

6.2 Means of Coordination

Dr. Gerkin trained in Pittsburgh and maintains ongoing collaborations with Dr. Aldrich, Mr. Omar, and Dr. Tripathy. 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 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 NSF II-0613404, CRCNS: Behaviorally Relevant Neuronal Modification during Postembryonic Development, which was funded from 10/06 through 9/10. This work examined how dendritic structure and synapse distributions follow functional architecture principles that relate directly to a neuron's individual behavioral requirements in holometabolous insects, such as Manduca sexta and Drosophila melanogaster. The modeling activities focused on flight motoneurons, which generally transform from tonically firing larval crawling motoneurons into phasically firing adult flight motoneurons. This ongoing work contributed to four publications so far [46, 47, 48, 49] with two more in review, as well as seven published abstracts [50, 51, 52, 53, 54, 55, 56], and 12 poster or oral presentations at conferences. It also provided partial funding for training opportunities for five graduate students and a postdoctoral researcher.

Jonathan Aldrich - Intellectual Merit: In his work on NSF CAREER grant CCF-0546550, "Lightweight Modeling and Enforcement of Architectural Behavior" (2006-2010, $500,000), Aldrich and his students developed a new approach to extracting run-time architectures from code and analyzing their behavior, producing 12 major conference[1] and journal publications and numerous workshop papers, including [57, 58, 59, 60, 61, 62, 63, 64, 65]. **Broader Impacts:** The NSF grant mentioned above, and 3 others, have contributed to the education of 11 Ph.D. students, including 4 women, and at least that many undergraduate or master's students. The grants have also supported the development and release of a number of open-source tools, including ArchJava [66], Plural [67], Crystal [68], and SASyLF [69]. All of these tools, in addition to educational methodologies developed in the grants, have been used in education at Carnegie Mellon and at other institutions. Recognition of the broader impacts of Aldrich's work on software architecture include the 2007 Dahl-Nygaard Junior Prize and an ICSE most influential paper award.

---

[1]note that in the field of computer science, major conferences are more prestigious than journals

References

[1] *http://www.crcns.org/*.

[2] *http://www.opensourcebrain.org*.

[3] *http://www.kaggle.com*.

[4] Jennifer Carpenter. May the best analyst win. *Science (New York, N.Y.)*, 331(6018):698–699, February 2011. PMID: 21311005.

[5] Renaud Jolivet, Felix Schürmann, Thomas Berger, Richard Naud, Wulfram Gerstner, and Arnd Roth. The quantitative single-neuron modeling competition. *Biological Cybernetics*, 99(4):417–426, November 2008.

[6] J J Hopfield and C D Brody. What is a moment? ”Cortical” sensory integration over a brief interval. *Proceedings of the National Academy of Sciences of the United States of America*, 97(25):13919–13924, December 2000. PMID: 11095747.

[7] *http://www.diademchallenge.org*.

[8] *http://www.the-dream-project.org/*.

[9] Kent Beck. *Test Driven Development: By Example*. Addison Wesley, 2003.

[10] *http://neuroml.org*.

[11] Padraig Gleeson, Sharon Crook, Robert C Cannon, Michael L Hines, Guy O Billings, Matteo Farinella, Thomas M Morse, Andrew P Davison, Subhasis Ray, Upinder S Bhalla, Simon R Barnes, Yoana D Dimitrova, and R Angus Silver. NeuroML: a language for describing data driven models of neurons and networks with a high degree of biological detail. *PLoS computational biology*, 6(6):e1000815, June 2010. PMID: 20585541.

[12] *http://www.neuron.yale.edu/neuron*.

[13] Karthik Ram. Git can facilitate greater reproducibility and increased transparency in science. *Source code for biology and medicine*, 8(1):7, 2013. PMID: 23448176.

[14] *http://www.djangoproject.com*.

[15] *http://github.com/cyrus-/sciunit*.

[16] *http://www.r-project.org/*.

[17] *http://www.mathworks.com/products/matlab/*.

[18] *http://opensourcebrain.org/projects/ca1pyramidalcell*.

[19] *http://www.scidash.org*.

[20] *www.sagenb.org*.

[21] *http://github.com/rgerkin/neurounit*.

[22] *http://www.neuroconstruct.org*.

[23] Padraig Gleeson, Volker Steuber, and R Angus Silver. neuroConstruct: a tool for modeling networks of neurons in 3D space. *Neuron*, 54(2):219–235, April 2007. PMID: 17442244.

[24] *http://senselab.med.yale.edu/modeldb*.

[25] Michael L Hines, Thomas Morse, Michele Migliore, Nicholas T Carnevale, and Gordon M Shepherd. ModelDB: a database to support computational neuroscience. *Journal of computational neuroscience*, 17(1):7–11, August 2004. PMID: 15218350.

[26] *http://github.com/rgerkin/neuroConstruct*.

[27] *http://neuralensemble.org*.

[28] *http://www.neuroelectro.org/*.

[29] SJ Tripathy, J Saviskaya, RC Gerkin, and NN Urban. NeuroElectro: a database describing the electrophysiology properties of different neuron types. *Neuroinformatics Meeting*, 2012.

[30] *http://bitbucket.org/rgerkin/neuroelectro_org*.

[31] *http://sed-ml.org/*.

[32] M Hucka, A Finney, H M Sauro, H Bolouri, J C Doyle, H Kitano, A P Arkin, B J Bornstein, D Bray, A Cornish-Bowden, A A Cuellar, S Dronov, E D Gilles, M Ginkel, V Gor, I I Goryanin, W J Hedley, T C Hodgman, J-H Hofmeyr, P J Hunter, N S Juty, J L Kasberger, A Kremling, U Kummer, N Le Novère, L M Loew, D Lucio, P Mendes, E Minch, E D Mjolsness, Y Nakayama, M R Nelson, P F Nielsen, T Sakurada, J C Schaff, B E Shapiro, T S Shimizu, H D Spence, J Stelling, K Takahashi, M Tomita, J Wagner, J Wang, and SBML Forum. The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics (Oxford, England)*, 19(4):524–531, March 2003. PMID: 12611808.

[33] Nikolaus Kriegeskorte, Marieke Mur, and Peter Bandettini. Representational similarity analysis - connecting the branches of systems neuroscience. *Frontiers in systems neuroscience*, 2:4, 2008. PMID: 19104670.

[34] TJ McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, 2(4):308–320, 1976.

[35] Michael Schmidt and Hod Lipson. Distilling free-form natural laws from experimental data. *Science (New York, N.Y.)*, 324(5923):81–85, April 2009. PMID: 19342586.

[36] *http://nipy.org/nibabel*.

[37] *http://neurohdf.readthedocs.org*.

[38] Cladius Ptolemy. *The Almagest*. 150.

[39] Johannes Kepler. *The Rudolphine Tables*. 1627.

[40] Nicolas Copernicus. *De revolutionibus orbium coelestium*. 1543.

[41] Johannes Kepler. *Astronomia Nova*. 1609.

[42] Galileo Galilei. *Siderius Nuncius*. 1610.

[43] Isaac Newton. *Philosophiae Naturalis Principia Mathematica*. 1687.

[44] Urbain Le Verrier. Lettre de m. le verrier à m. faye sur la théorie de mercure et sur le mouvement du périhélie de cette planète. *Comptes rendus hebdomadaires des séances de l'Académie des sciences (Paris)*, 49:379–383, 1859.

[45] Albert Einstein. The foundation of the general theory of relativity. *Annalen der Physik*, 49(7):769–822, 1916.

[46] S Hohensee, W Bleiss, and C Duch. Correlative electron and confocal microscopy assessment of synapse location in the central nervous system of an insect. *Journal of Neuroscience Methods*, 168:64–70, 2008.

[47] M Meseke, JF Evers, and C Duch. Developmental changes in dendritic shape and synapse location tune single neuron computations to changing behavioral functions. *Journal of Neurophysiology*, 102:41–58, 2009.

[48] MA Herrera-Valdez. Membranes with the same complements of ion channesls but different excitabilities. *PloS ONE*, 7(4):e34636, 2012.

[49] MA Herrera-Valdez, EC McKiernan, SD Berger, S Ryglewski, C Duch, and S Crook. Relating ion channel expression, bifurcation structure and diverse firing patterns in a model of an identified motor neuron. *Journal of Computational Neuroscience*, 34(2):211–229, 2013.

[50] AB Jennings, SM Crook, C Duch, and S Ryglewski. Mathematical models of octopaminergic dorsl unpaired median neurons. *Society for Neuroscience Abstracts*, (536.20), 2007.

[51] MA Herrera-Valdez, SD Berger, C Duch, and SM Crook. Predicting changes in neuronal excitability type in response to genetic manipulations of k+ channels. *BMC Neuroscience*, 10((Suppl 1)):P301, 2009.

[52] SD Berger, MA Herrera-Valdez, C Duch, and SM Crook. Passive current transfer in wildtype and genetically modified drosophila motoneuron dendrites. *BMC Neuroscience*, 10((Suppl 1)):P346, 2009.

[53] MA Herrera-Valdez, SD Berger, C Duch, and SM Crook. Differential contribution of voltage-dependent potassium channels to neuronal excitability. *BMC Neuroscience*, 11((Suppl 1)):P159, 2009.

[54] A Smith, M Cruz-Aponte, EC McKiernan, MA Herrera-Valdez, SM Crook, and MA Herrera-Valdez. Differential contributin of a-type potassium currents in shaping neuronal responses to synaptic input. *BMC Neuroscience*, 12((Suppl 1)):P147, 2011.

[55] SD Berger, SM Baer, and SM Crook. Estimation of electrical properties of dendrites with branches using a continuum modeling formulation. *Society for Neuroscience Abstracts*, page 340.01, 2012.

[56] SD Berger, SM Baer, and SM Crook. A continuum approach to model neurites/dendrites with emerging subtrees. *BMC Neuroscience*, 14((Suppl 1)):P73, 2013.

[57] Marwan Abi-Antoun, Jonathan Aldrich, and Welsey Coelho. A Case Study in Re-engineering to Enforce Architectural Control Flow and Data Sharing. *J. Systems & Software*, 80(2):240–264, 2007.

[58] Kevin Bierhoff and Jonathan Aldrich. Modular Typestate Checking of Aliased Objects. In *Proc. Object-Oriented Programming, Systems, Languages, and Applications*, 2007.

[59] Marwan Abi-Antoun, Jonathan Aldrich, Nagi Nahas, Bradley Schmerl, and David Garlan. Differencing and Merging of Architectural Views. *Automated Software Engineering*, 15(8):35–74, 2008.

[60] Marwan Abi-Antoun and Jonathan Aldrich. Ownership Domains in the Real World. In *Intl. Workshop on Aliasing, Confinement and Ownership in Object-Oriented Programming (IWACO)*, pages 93–104, 2007.

[61] Marwan Abi-Antoun and Jonathan Aldrich. Compile-Time Views of Execution Structure Based on Ownership. In *Intl. Workshop on Aliasing, Confinement and Ownership in Object-Oriented Programming (IWACO)*, pages 81–92, 2007.

[62] Marwan Abi-Antoun and Jonathan Aldrich. A Field Study in Static Extraction of Runtime Architectures. In *Workshop on Program Analysis for Software Tools and Engineering (PASTE)*, pages 22–28, 2008.

[63] Marwan Abi-Antoun and Jonathan Aldrich. Static Extraction of Sound Hierarchical Runtime Object Graphs. In *Workshop on Types in Language Design and Implementation (TLDI)*, pages 51–64, 2009.

[64] Marwan Abi-Antoun and Jonathan Aldrich. Static Extraction and Conformance Analysis of Hierarchical Runtime Architectural Structure using Annotations. In *Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, 2009.

[65] Marwan Abi-Antoun and Jeffrey M. Barnes. Analyzing Security Architectures. In *Automated Software Engineering*, 2010.

[66] Jonathan Aldrich. ArchJava. *http://www.archjava.org/*, 2003.

[67] Plural Project. Pluralism: Modular object protocol checking for Java, 2009. *http://code.google.com/p/pluralism/*.

[68] PLAID Research Group. The Crystal Static Analysis Framework, 2009. *http://code.google.com/p/crystalsaf*.

[69] Jonathan Aldrich. SASyLF proof assistant, 2009. *http://www.cs.cmu.edu/~aldrich/SASyLF/*.

*Todo list

Remove bibliography with nobibliography tag before submission