REENGINEERING SWMM/EPANET   
USER INTERFACE APPLICATION   
SOFTWARE ARCHITECTURES

**QAPP-16**

Revision 0

*prepared for*

U.S. Environmental Protection Agency

Office of Research and Development

26 West Martin Luther King Drive

Cincinnati, Ohio 45268

November 2015



REENGINEERING SWMM/EPANET   
USER INTERFACE APPLICATION   
SOFTWARE ARCHITECTURES

EPA Contract #GS-10F-0041X

**RSI-QAPP-16**

Revision 0

*by*

AQUA TERRA Consultants

2685 Marine Way, 1314

Mountain View, California 94043

*prepared for*

U.S. Environmental Protection Agency

Office of Research and Development

26 West Martin Luther King Drive

Cincinnati, Ohio 45268

November 2015

**Prepared By:**

Paul Hummel Date

Project Manager, AQUA TERRA Consultants

**Approved By:**

John Imhoff Date

Quality Assurance Officer, AQUA TERRA Consultants

**Approved By:**

Robert Janke Date

Contracting Officer’s Representative, EPA/NHSRC

**Approved By:**

Michael Tryby Date

Cocontracting Officer’s Representative, EPA/NHSRC

**Approved By:**

Sini Jacob Date

Contracting Officer, EPA/OARM/OAM/HPOD/IRMPSC

**Approved By:**

Ramona Sherman Date

Quality Assurance Manager, EPA/ORD/NHSRC

TABLE OF CONTENTS

1.0 DISTRIBUTION LIST 1

2.0 PROJECT ORGANIZATION 2

3.0 PROJECT BACKGROUND AND OBJECTIVES 4

4.0 PROJECT DESCRIPTION 5

5.0 Quality Assurance Components for Software Development 6

5.1 Coding Style Standard 6

5.2 Documentation Standard 6

5.3 Testing Standard 7

5.3.1 Levels of Testing 7

5.3.2 Objectives of Testing 8

5.3.3 Relationship Between the Testing Process and the Development Process 8

5.3.4 Software Testing Process 9

5.4 Source Code Management 10

5.5 Management of Ancillary Project Materials 11

5.6 Software Build, Packaging, and Delivery 11

5.7 Support 12

6.0 TASK EXECUTION PLAN 13

6.1 Software Design Process 13

6.2 Project Management Structure 13

6.3 Project Management Controls 15

6.4 Communication and Meetings 18

6.5 Project Risk Management 18

6.6 Deliverables and Schedule 19

7. 0 References 20

LIST OF TABLES

TABLE PAGE

6‑1 Deliverable Schedule 19

LIST OF FIGURES

FIGURE PAGE

2‑1 Quality Assurance Organizational Diagram 3

6‑1 Project Management Organization Chart 14

6‑2 Sample Screen Shot of the Trello Management Tool 17

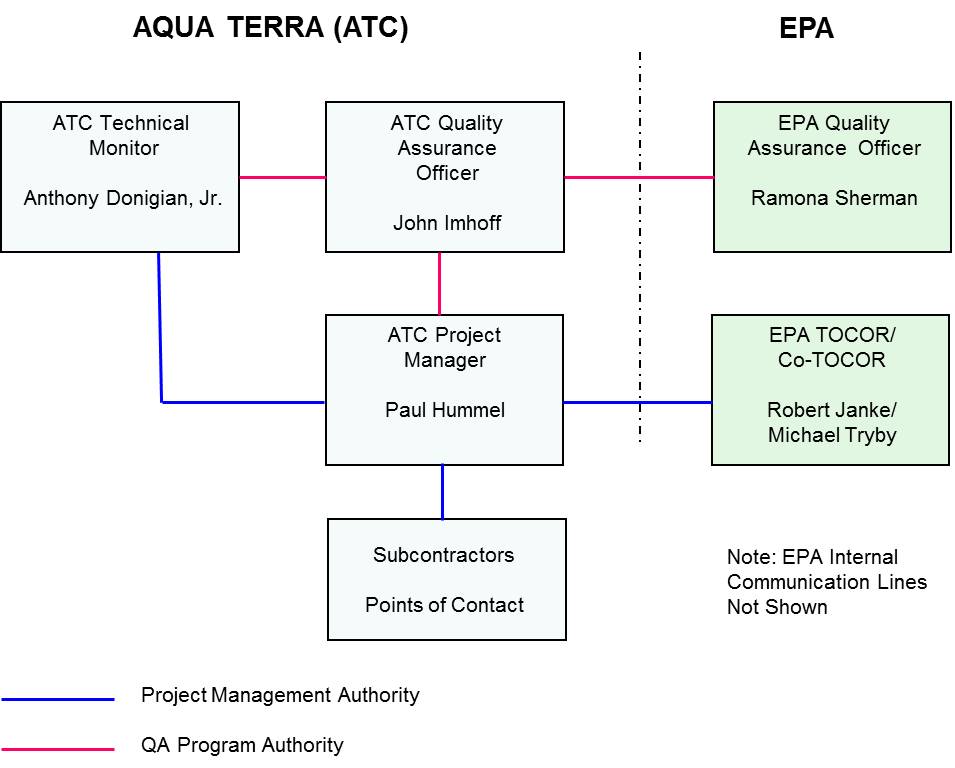
# 0 DISTRIBUTION LIST

|  |  |
| --- | --- |
| **Name and Title** | **Organization and Contact Information** |
| **U.S. Environmental Protection Agency** | |
| Robert Janke Contracting Officer’s Representative | Office of Research and Development, NHSRC Mail Code: NG16  26 West Martin Luther King Drive  Cincinnati, Ohio 45268 513-569-7160 (voice) *janke.robert@epa.gov* |
| Michael Tryby Cocontracting Officer’s Representative | Office of Research and Development, NHSRC Mail Code: 689 26 West Martin Luther King Drive  Cincinnati, Ohio 45268 513-569-7559 (voice) *tryby.michael@epa.gov* |
| Ramona Sherman Quality Assurance Officer | Office of Research and Development, NHSRC Mail Code: NG16  26 West Martin Luther King Drive  Cincinnati, Ohio 45268 513-569-7640 (voice) *sherman.ramona@epa.gov* |
| Sini Jacob Contracting Officer | Mail Code: 3803R  William Jefferson Clinton Building  1200 Pennsylvania Avenue, N. W.  Washington, DC 20460 202.564.3054 (voice) *jacob.sini@epa.gov* |
| **AQUA TERRA Consultants** | |
| Paul Hummel Project Manager | 150 E. Ponce de Leon Avenue, Suite 355 Decatur, GA 30030 404.378.8337 (voice) 404.378.8332 (fax)  *paul.hummel@respec.com* |
| John Imhoff Quality Assurance Officer | 735 Main Street P.O. Box 323 Ouray, CO 81427 970.325.4283 (voice)  970.325.4328 (fax) *john.imhoff@respec.com* |

# 0 PROJECT ORGANIZATION

This section discusses all important intramural and extramural project personnel and shows the relationship between the development team and the personnel responsible for quality assurance (QA) and testing. The U.S. Environmental Protection Agency (EPA) assigned Ms. Ramona Sherman as the Quality Assurance Officer, Mr. Robert Janke as the Task Order Contracting Officer’s Representative (TOCOR), and Mr. Michael Tryby as the Co-TOCOR.

Mr. Paul Hummel will be the Project Manager for AQUA TERRA Consultants, and he is responsible for providing technical direction to staff and interacting with the EPA TOCOR and Co-TOCOR. Mr. Anthony Donigian will serve as AQUA TERRA’s Technical Monitor for this project and perform selected administrative and review tasks. Mr. Brian Bicknell will be assisting Mr. Donigian. Mr. John Imhoff, AQUA TERRA’s Quality Assurance Officer, will supervise quality assurance activities. Other AQUA TERRA staff will be available to assist Mr. Hummel in this project, including Mr. Mark Gray, Mr. Paul Duda, and Dr. Tong Zhai. AQUA TERRA will be assisted by subcontractors Computational Hydraulics International (CHI) and HydroGeoLogic, Inc. (HGL) to aid in technical design and assistance and in testing aspects of software development. Mr. Rob James will serve as the Point of Contact (POC) for CHI and Mr. Dua Guvanasen will serve as the POC for HGL. Figure 2-1 illustrates the organizational diagram for QA.

Figure 2‑1. Quality Assurance Organizational Diagram.

# 0 PROJECT BACKGROUND AND OBJECTIVES

The purpose of this project is to obtain software development and support services for the EPA’s collection system and water distribution system simulation products, the Storm Water Management Model (SWMM) and software that models water distribution-piping systems (EPANET). The project will focus on the design of a modular and extensible User Interface (UI) application software architecture for SWMM and EPANET. The modular and extensible UI application software architecture will enable deployment of new application features created by the EPA, third-party developers, and end users employing application “plug-ins.” The project team will employ software development best practices related to design, development, testing, documentation, delivery, installation, and maintenance to ensure cost effectiveness of this project and performance of the resulting software applications.

Major objectives of this project are to reengineer the SWMM and EPANET UI applications architecture to be modular, which will allow for plug-in and scripting support. The EPA cannot anticipate all of the features application users will need in the future, so the applications will need to allow for change and adaptation. The newly designed modular plug-in architecture will be flexible and allow the plug-ins to be developed in C++ or Python. High-level scripting support within the application will allow users to more easily manage input data, drive specialized simulations, and analyze output data. The modular and extensible design will encourage users to develop solutions independently, with a mechanism to share these solutions with the broader community.

Further details about the project background and detailed descriptions of the tasks to be performed in this project can be found in the project’s Statement of Work (SOW). The following sections of this Quality Assurance Project Plan (QAPP), including the Task Execution Plan, describe quality assurance measures to be used by the project team while carrying out the tasks.

# 0 PROJECT DESCRIPTION

As described in the SOW, this project has the following five primary tasks:

1. Predevelopment
2. Active development
3. Product delivery
4. Computational engine support
5. Postrelease support of SWMM/EPANET UI applications.

Predevelopment tasks will consist of developing a QAPP (this document), an Application Features Requirements Document (AFRD), and a software application architecture design memorandum. As the project team embraces the agile development methods preferred for this project, developing well-designed predevelopment products is critical. The AFRD will be essential in establishing short- and long-term goals during the development process. The architecture design memorandum will provide the overall guidance needed to ensure that quality, useful products are produced and the client’s expectations are met.

As noted in the solicitation’s SOW, the EPA (along with other government agencies) has embraced the techniques of agile software development. For this project, active development will be carried out under a set of predefined Minimum Testable Products (MTPs). Using the AFRD as a guide, MTPs will be defined under close coordination with EPA staff. A minimum of three MTPs are anticipated for this project. Each MTP will be executed through a series of iterative development sprints (i.e., short [2 – 3 weeks] periods of development and testing to complete a set of goals).

The MTPs are anticipated to be completed during the first year of the project. Upon completion of the MTPs, the reengineered, fully functional SWMM and EPANET software will be delivered to the client along with all development and application resources. While extensive testing of the code will be performed under MTP development, further testing of the full UIs will be performed during the second year of the project. When the testing is completed, final versions of the SWMM and EPANET UIs will be formally released to the user community. The distribution packages, along with all source code, documentation, and other development resources, will be delivered to the EPA.

The project team’s technical approach for computational engine support will involve selecting, in collaboration with the EPA TOCOR, a compiler, code editor, and source-code debugger for each target operating system. The computation engine codes will then be ported to this selected development environment. The project team will work in coordination with the EPA to develop a full suite of test runs that exercise a broad and deep range of each model’s functionality.

The project team will provide end-user support when the two programs have been released. This support will include addressing bug reports, developing frequently asked questions (FAQs) to address repeated technical questions, and developing technical notes to provide an in-depth response to a particular issue. Update patches will be developed as needed along with updates of unit and regression tests and system and end-user documentation.

# 0 Quality Assurance Components for Software Development

The Quality Assurance/Quality Control (QA/QC) goals for this software development project include the following:

* Objectivity—each step in the software development cycle should use methods that are explicitly stated and adhered to
* Thoroughness—all elements of the study should be carried out and documented in a thorough manner
* Consistency—all work should be performed and documented in a consistent manner
* Transparency—the programming code, testing scripts, and documentation should clearly describe the assumptions and methods used and verify the capabilities specified in the functional requirements.

Quality assurance, as it relates to developing environmental software is achieved by adhering to a number of practices that are described in various EPA guidance manuals [EPA, 2003; EPA 2002; Council for Regulatory Environmental Modeling, 2009]. Guidelines and procedures are needed to govern software development. Sections 5 and 6 describe the specific QA activities relevant to this project’s software development.

## Coding Style Standard

We will use the Python Enhancement Proposal (PEP) 8 coding standard [Van Rossum, 2001] will be used for this project. This is Python’s “gold standard,” which is created, maintained, and widely used by the Python development community. The “pep8” utility will be used as a step of the automated testing process to check adherence to this standard. Programmers are expected to make their code follow PEP 8 before it is committed to the repository. If automated testing determines that the code does not follow PEP 8, it will be brought to the programmer’s attention and will be prioritized for fixing as a failing unit test. Python code created outside this project, such as by third-party, open-source projects, will likely follow this standard because it is popular, but to preserve the ability to synchronize with newer versions, third-party code will not be changed to enforce this standard.

## Documentation Standard

Internal code documentation, in the form of comments and carefully chosen module, method, class, and variable names, will be required for all source code. Creating and editing source code includes creating and editing internal documentation as needed for it to reflect the functionality of the code. We will use the Doxygen tool to create system documentation based on the source code and internal code documentation. Doxygen is the standard tool for generating documentation, including Unified Modeling Language (UML) diagrams, from annotated C++ and it now also supports Python. We will ensure that our coding and internal documentation style is compatible with Doxygen. Any individual who finds internal documentation that needs improvement for any reason is encouraged to file a report in the project issue tracking system to be prioritized like any other issue needing attention. Unit testers are encouraged to review internal documentation for completeness and accuracy when creating and updating unit tests. System document­ation will describe the SWMM and EPANET objects so their structures can be understood by the open source development community.

User interface documentation for SWMM and EPANET will be provided as searchable hyperlinked help. The new UI documentation for both programs will follow closely the existing manuals and will be updated to reflect the changes to the system. User and system documentation will be reviewed and updated before delivering each MTP.

## Testing Standard

The following sections describe the levels at which testing is conducted, the objectives of testing, the relationship between the testing and development process, and the testing process for components of the system. The principles that guide these testing procedures are grounded in standards [EPA, 1999] adopted by the EPA and have been embraced by the project team through numerous software development projects. The standards have been updated here to meet the specific demands of this project.

### Levels of Testing

Software testing will be performed at both the unit and system levels. **Unit testing** evaluates individual components in isolation from other components. Public methods of a component are those that are available to other components. Public methods are also called the API (Application Programming Interface) of a component. Each public method of a component must have an associated unit test method. Python’s built-in “unittest” framework will be used, which is available to all Python programmers. Coverage of unit tests will be evaluated during unit testing using Python’s “coverage.py” to verify that most lines of code are exercised by unit tests. It is expected that some code cannot be covered by tests, such as when a potential error condition is always prevented by other code but is still checked for or when file system errors or race conditions are checked for but are impossible to engineer into a test. Coverage gaps that are detected will be included in the testing report and will be evaluated to determine whether or not additional tests are needed.

In addition to unit testing, the performance of groups of components functioning together will be tested. **System testing** includes integration testing (data communication between components), functional testing (the overall performance of the system), and regression testing. When a problem is fixed, a regression test is added. Passing a regression test demonstrates that the problem has been fixed. It is called a regression test because continuing to test for it in the future provides detection of “regressing”—reintroducing the same problem later. System tests that can be automated will be incorporated into the same framework used for unit testing. Some system tests are difficult or impossible to automate and instead will be accomplished by a tester performing a set of actions and verifying the results manually.

### Objectives of Testing

The following three main objectives are associated with testing:

1. *Uncover errors in requirements, design, and programming.* Software developers may have to revise requirements documentation, design documentation, and program code implementation to resolve errors.
2. *Identify or confirm limitations of the system.* Software developers will need to document these limitations (perhaps in a revision to the design documentation) for future reference. Software developers will also have to eliminate limitations that inhibit the required functionality of the software.
3. *Build confidence in the capabilities of the system.* After testing, the tester should be confident that the software meets its requirements. Testers shall keep documentation to support this conclusion.

It is important to keep in mind that complete testing of any software is not possible. For software of even modest complexity, the set of all possible cases (i.e., all possible input data and combinations of input data items) is essentially unlimited. Therefore, the testing process will be thought of as a type of statistical analysis. The tester attempts to:

1. Select a representative sample of cases from the set of all possible cases.
2. Evaluate the software's performance on the sample cases.
3. Make an inference about the software's performance on the complete set.

This last step involves uncertainty because showing that software works properly on the sample cases does not mean it will work properly on all possible cases. However, successful testing does provide confidence that the software meets its functional requirements and will perform acceptably on most of the possible cases.

Quality testing requires an in-depth knowledge of the system being tested. This knowledge includes understanding the input data, the solution technique implemented by the software, and what the output data should look like. Quality testing can be done independently from the software developers. Knowledge of the software developer's thought process is not necessary to verify that the code works properly and logically.

### Relationship Between the Testing Process and the Development Process

The testing process runs parallel to the development process. Both processes begin by identifying requirements. The requirements are reviewed for testability and clarity and to ensure that they will meet the needs of the user. The software design is also reviewed for testability and clarity, as well as to ensure that all requirements are addressed.

Before each code change is considered complete and eligible for inclusion in a release, unit testing of this code must be complete. The testing report for each release is expected to be “all green” with all tests passing for the version of the code included in the release. Good modular design and using a source-code repository simplify the process of omitting any changes that are not ready for release. We are not pursuing a test-driven development model where tests are always written before code, but when a test is written it may test more features or conditions than have already been implemented. If a test fails, but only for a situation the software is not yet required to handle, the code change can be considered complete. In such a case, the failing test result will be reflected in a release note as a “known issue” of the release.

### Software Testing Process

A “first draft” set of unit tests for each newly added public method is the responsibility of the developer who creates the method. These tests must be complete in the sense that all of the required test routines exist, but a first draft does not need to be fully implemented. Any unfinished tests must be designed to fail so a testing report will show that they need attention. All related tests must pass before the change is considered complete. All tests will be reviewed by an independent party who is responsible for running tests and generating testing reports. Tests may be implemented and edited by any developer as assigned by the scrum work flow.

To compile a list of needed system tests, each requirement in the Application Features Requirements Document (AFRD) will be examined to determine which automatable or manual methods can be used to test whether or not the requirement has been met. Each interaction between software components will also be examined for testing opportunities. As each test is first implemented, it will by default fail with an indication that the test is incomplete. The test will only be allowed to succeed (by removing the default “test incomplete” failure) when the test is fully implemented. As the AFRD and interactions between components are updated, the suite of system tests will be updated as well.

Working iteratively, any failing test results will be added to the scrum backlog. Tests that fail because the test is not yet completely implemented will be tagged as such. Once they are in the backlog, failing tests will be prioritized and addressed in a subsequent sprint by appropriate revisions to code, design, requirements, documentation, and/or testing.

Automation of tests, including unit, coverage, system, and code style checks, will be implemented by custom Python scripts that will be maintained in GitHub alongside the test routines. These scripts will allow all of the tests for the whole system to be run with one command, which generates a complete automated testing report. Because running all tests will take a long time, the scripts will also support running only a subset of the tests for faster iterative development. This way, when a programmer is working toward making a particular routine or module pass its tests, a smaller scope of only one test or only the tests for a particular module can be run.

Each manual (nonautomated) test will be implemented as a document containing instructions and expected results. The tester will follow the instructions and record in the manual testing report whether or not the expected results were seen. Some manual tests will be focused on one feature. Other manual tests will cover an entire work flow and will have several steps and several intermediate results to check for.

All automated and manual tests will be run on the release version of the system before each release and all test results will be provided with each release.

## Source Code Management

GitHub is a Distributed Version Control System with many features that support a multideveloper team as well as the capability for others outside the core team to make their own changes, which may or may not get folded back into the original project. The primary operations of interest on GitHub are branch, commit, pull, tag, and fork. (All of these terms are used both as an action for performing the operation and as a result when referring to the record of the operation in the repository.) A developer within the core project team creates a **branch** when starting work on a new change. Within a new feature branch, changes are free to be experimental and untested because they only affect others currently working on that branch. A developer is encouraged to **commit** (save to GitHub) incremental changes frequently (daily or whenever a particular aspect of a change is complete) so other team members can see what they are working on and minimize conflicting changes to the same code. If a developer is working on a feature that takes a long time to implement and does not wish to commit a half-finished or temporarily broken version to GitHub, they are encouraged to commit to their local repository and coordinate with other developers daily and are required to commit to GitHub before the end of the sprint. When committing, the developer writes a comment describing the purpose of the changes. If more than one developer is working on the same branch, frequent commits assist with coordination and each commit requires care to avoid causing problems for other developers. A key feature of a commit is that it can be undone if it is found that it introduced a bug or if a different approach is pursued. A **pull** request is made when a developer believes the changes in a branch are complete. This initiates a discussion with other team members about whether or not everyone agrees that the branch is fully tested and ready for the merge. After QA is performed on the branch and team members agree, the changes in the branch are pulled into the master branch. This will generally occurs at the end of each sprint. Whenever the master branch is updated, at least one GitHub issue tracker item is usually affected. Keywords in a commit message can be used to automatically link an issue with its solution and mark it as solved when the code is accepted into the master branch. A **tag** marks a particular revision in the repository, usually to record the exact version of the source code that was used to build a particular release package. The GitHub feature of creating a release will be used, which includes a tag along with an indication of whether or not the release is stable. A release provides a convenient way for the public to download the source code and install a release. Each release will be named using the date when it is made and something memorable about the release such as the major feature added; for example, “2015-10-14: Integrated Map.”

Individuals outside the project team do not have write access to change code in the repository themselves but, in a public repository, they can see everything and are encouraged to become involved by submitting issues and commenting in the issue tracker. Outside developers can easily create their own **fork**, which is an efficient copy of the whole project. Within their fork, they can create their own branches and commit changes. Outside developers may simply want to maintain customized versions for themselves or they can use a pull request to offer their changes to the original project. If we find their code useful and we can make it meet our code style and testing criteria, then it will be added to the repository. Outside developers who prove valuable to the project may be added to the team and given direct access to the repository if preferred.

If an existing third-party GitHub project is modified in this effort, we will make a GitHub fork of that project to contain our changes. This will allow us to easily incorporate new changes made in the original project. If we make improvements of interest to the original project, this strategy will easily support offering our changes to them.

For all of the GitHub repositories and forks created for this project, in addition to the program source code and supporting build files, all testing code and reports and all supporting materials, such as design documents, user and developer reference documents, advanced application manuals, examples, and tutorials, will be maintained on GitHub where every team member will have read access. Write access may be limited to those responsible for maintaining certain sections of the repository, but the TOCOR will have complete administrative access including write access and team membership control.

## Management of Ancillary Project Materials

As stated above, all testing code and reports and all supporting materials such as design documents, user and developer reference documents, advanced application manuals, examples, and tutorials will be maintained on GitHub where every team member will have read access. If the EPA requests that any files not be available on GitHub, we will honor that request and deliver those project materials separately.

## Software Build, Packaging, and Delivery

Software building and packaging will be automated by custom Python scripts. A particular version of the source code will be selected by tag or revision and that version will be pulled from GitHub and built into an install package. We plan to use the cross-platform PyInstaller for packaging. This will allow us to reuse some packaging effort across platforms, but development of the particular packaging technique for each platform will be a separate task. Packaging for Windows, the primary target platform, will be developed and refined first. Packaging for Mac OS X and Linux will be delivered later, as prioritized in consultation with the EPA and is not expected to be a priority in the first year.

Delivery will be accomplished by adding install packages to a GitHub release where users can access them. Installation instructions will be included in each GitHub release. Before release, each install package will be tested on its target platform. The number of possible machine configurations for each platform is unlimited. When creating a release, the team will decide what configurations to test first; for example, “Windows 7, 64-bit, non-administrator account.” Testing of other configurations will be added as important configuration differences are found by internal testing and by user reports.

It may be preferable to share a new design with people outside the team, for example with interested users, before the design is completely implemented or tested to receive early feedback. Such a preview may be a design sketch or a nonfunctional mockup or a partially functional demonstration. If such a “demo version” is created, it will not be considered a release package, it will not go through the full testing process, and it will be appropriately labeled to limit user expectations.

## Support

As a general approach to support, we will work to minimize support required for the code in the public repository. Two general approaches are used to achieve this goal. First, we require that the developers ensure that all code necessary to build the projects is committed to the repository. In other words, at all times, any programmer should be able to download the source code of the latest release and build the projects without having any unresolved components. It is up to the developers to provide complete source-code commits. Second, notes within the commits to the repository will include explanations of what the commit contains, how it has been tested, and known issues. This approach further minimizes the inappropriate expectation of someone downloading the code and expecting it to do more than has been implemented. For instance, if an “edit” form has been committed but the code to return parameter values from the form back into the object has not yet been implemented, this should be noted in the commit message.

Despite these efforts to minimize the need for support, we recognize that during the course of using the new UIs and their supporting code, users will likely report program bugs and make feature requests. Users and outside developers will be encouraged to use the GitHub repository to report all issues. The GitHub issue system will be integral to the development process so that issue reports will be seen by the team. In consultation with the TOCOR, the team will reply to the issue report, decide whether or not further action is needed, and prioritize action as with all tasks. Feature requests submitted on GitHub will be forwarded to the TOCOR for further consideration.

# 0 TASK EXECUTION PLAN

The Task Execution Plan (TEP) is extractable from the QAPP as a governing document that establishes the means to execute, monitor, and control the project tasks. The plan is the main communication vehicle to ensure that everyone is aware and knowledgeable of project objectives and how they will be accomplished. The TEP is the primary technical roadmap between the EPA and the project team. Project objectives are derived from the EPA’s Statement of Work, and the EPA and all key personnel on the project team assist in developing the TEP. The plan will be a living document and will be updated to describe current and future processes and procedures throughout the project. The TEP also contains the scheduling component that was previously a separate component of EPA software design and development QAPPs.

## Software Design Process

During proposal development for this project, the project team investigated a variety of alternatives for the reengineered SWMM/EPANET software architectures. This included consideration of the primary development languages, plug-in architecture issues, and the role of GIS. Discussions with EPA staff have helped clarify decisions related to these issues.

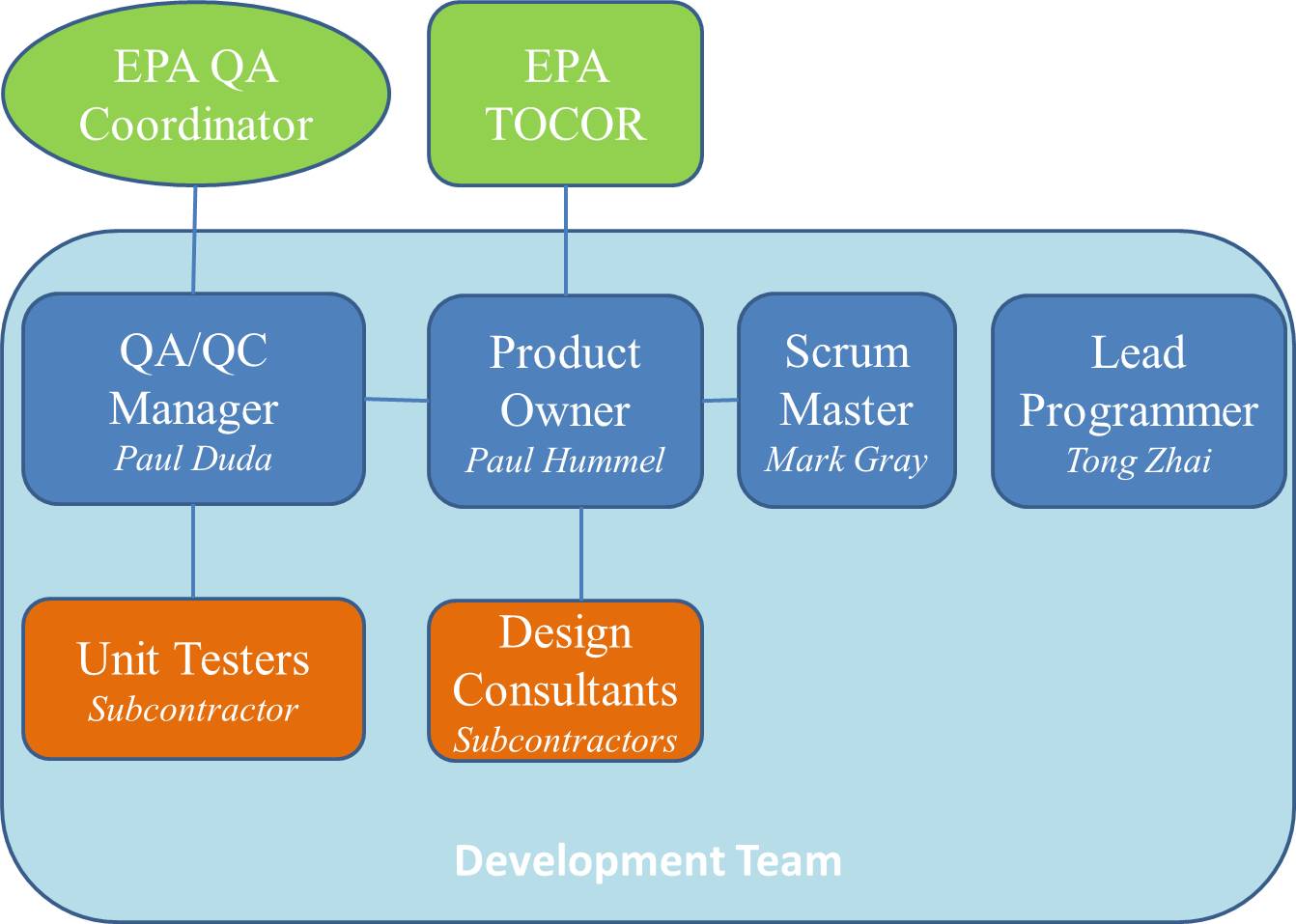
To fully clarify the system design, a detailed memorandum will be developed that describes all facets of the software architecture for this project. Decisions related to all aspects of software design for this project will be made through close coordination with the EPA. Sharing knowledge and experience between the EPA and the project team will be essential to establishing the best software architecture and development process for this project.

## Project Management Structure

The key to the successful operation of the project team is the structure and functioning of the **Project Management Plan (PMP)**. The PMP must be designed to establish clear lines of communication, effective technical coordination, efficient technical and financial reporting systems, and up-to-date project cost-accounting procedures. This section describes the PMP we have developed and refined over time to meet these criteria. The plan includes a project management approach and tools that are customized to meet the requirements of an agile programming environment, combined with our standard practices for ensuring cost control and quality control. The project team ensures client satisfaction by producing the highest quality services and products within specified schedule and budget constraints.

Our project team consists of AQUA TERRA Consultants as the prime contractor and Computational Hydraulics International and HydroGeoLogic, Inc. as subcontractors. The proposed internal program management organization is shown in Figure 6-1.

As prime contractor, AQUA TERRA will assume overall project management, supervision of subcontractors, and primary communication with the EPA Contracting Officer and Project Officer. The administrative line of communication between AQUA TERRA and its subcontractors (CHI and HGL) will be between AQUA TERRA’s project manager and the designated POC for each subcontracting firm as specified in section 2.0.

Figure 6‑1. Project Management Organization Chart.

The **development team** is the group of people that will design, program, test, and document to achieve the goals of the project. Mr. Paul Hummel, Mr. Paul Duda, Mr. Mark Gray, and Mr. Tong Zhai will be the core development team for this project. Subcontractors will also be members of the development team.

We will use scrum, a popular agile project management methodology, for this project. Scrum defines three core roles for people to fill: scrum master, product owner, and development team. These roles and individuals nominated for each role are described below. The scrum process we will follow is outlined below, which revolves around the list of tasks to be completed (the backlog.)

The **scrum master** has training in the scrum methodology, is the facilitator of meetings, and keeps the team following the scrum process and focused on making progress on the project. When an obstacle to progress is found, the scrum master works to find a solution. If an immediate solution is not found, the list of tasks provides the team with other tasks that are ready to be worked on so nobody is idle. Mr. Mark Gray will serve as the scrum master for this project.

The **product owner** is the primary point of contact with the client and represents the client’s interests in team meetings. The product owner’s role is to prioritize items that the team will work on based on knowledge of the client’s goals and the team’s current status and estimates. When a significant event occurs, such as reaching a milestone or making a release, the product owner communicates these events directly to the client and records them using the appropriate GitHub tools including updating the GitHub Pages project description and using the “Create a new release” feature. A very involved client may choose to follow team progress as closely as they wish by watching the team’s online support and communication systems (such as GitHub and Trello) and attending (remotely or in person) any meetings, but a client is unlikely to want to take on the full-time role of product owner. Mr. Paul Hummel will serve as the product owner for this project.

The **QA/QC manager** is responsible for overseeing the testing process during software development. It is essential for those writing and performing the unit tests to be involved in the development process. Thus, a key part of the manager’s job will involve coordinating current development activities with the testers. The manager will coordinate with the testers throughout development sprints to assist in unit test development and to assess unit test completion. For this project, Mr. Paul Duda will serve as the QA/QC manager.

The **lead programmer** is responsible for software development and maintaining code functionality and quality, which includes coding and code review to ensure functional achievements and adherence to coding standards adopted for this project. Additionally, this role oversees project documentation of both source code and the user interfaces following the practices described in Section 5.2. Mr. Tong Zhai will serve as lead programmer for this project.

Subcontractor roles will primarily focus on architecture and UI design assistance, computational engine support, and developed software products testing. CHI has domain expertise for SWMM as well as experience in using EPANET. As such, CHI will provide up-front insight in designing the new architecture for SWMM and EPANET to fully accommodate its current functionality. Further, because CHI serves as the foremost host for SWMM user community discussions, they are also very well informed for anticipating the most likely directions for SWMM enhancement. CHI’s contributions to the project will continue in a review capacity throughout the active development phase and will also involve supporting the SWMM and EPANET computational engines.

HGL will provide the project team with well-developed skills in the project’s development environment along with a broad range of knowledge and experience in innovative information technology. They will serve as technical advisors in designing the architectural framework for the new products. Additionally, HGL provides a broadened range of options for assigning required testing functions to personnel that are completely independent of the code designers/programmers. Under the supervision of the QA/QC manager, HGL staff will be integrated into the software development testing process. While they will focus on performing tests and reporting results, they will be involved in the overall development process to fully understand the functional requirements to be assessed in the tests.

## Project Management Controls

This project is first divided into MTPs as discussed elsewhere. Each MTP is then divided into multiweek **sprints**. During each sprint, the team works on a specified set of user stories, which are distinct software components that perform testable tasks. The team is self-organizing and may decide to change the process as needed to best support the project. While 3-week sprints are anticipated at the project start, the team may decide that for a particular phase of the project with many small user stories, sprints should be shortened to 1 or 2 weeks. Each sprint begins with a planning meeting where the set of user stories to be worked on during the sprint is selected from the backlog. Details of these items are added as needed to turn them into concrete tasks, perhaps breaking one item expressed as a user story into more than one developer work item, each with a rough estimate of how demanding they will be. The goal is to plan enough work in enough detail for everyone during the whole sprint while not focusing on too much detailed planning of things that will happen much farther in the future, because those details are likely to change. Having a little excess work planned in the sprint will ensure that there is a “stretch goal” to work on if a task is completed more quickly than expected or if something blocks further work on a task. At the end of a sprint, the team meets to review what was completed and what was not completed, what went well and what did not, and whether or not some change could be made to make the next sprint process smoother. Items that are not complete at the end of a sprint are edited to reflect their level of completion and are moved into the backlog. These items will then be prioritized alongside other work items being considered for the next sprint. At the end of a sprint, some newly complete features will be shared. The team will plan a software release, a client demonstration, and a blog post or project description update to share what is newly complete. For an intermediate sprint that does not end with an MTP, the team will choose whether or not a release and demo are worthwhile.

Agile development methods emphasize showing only new functionality that is completely finished and thoroughly tested. This is an appropriate protocol for committing code to the main branch or building a production version of the software, but, in our experience, it can be beneficial to show new functionality at every stage. The earlier a customer can see what is being proposed, the sooner they can direct us toward the preferred result. This means that, in addition to showing finished features, work in progress will also sometimes be presented for early feedback. For UI components, this means sharing and getting feedback on early versions such as a rough sketch of a new window or a mockup that implements little or none of the planned functionality. When working on creating the data model, this means sharing and discussing an early “skeleton” version of the model before all fields and supporting methods are finalized, implemented, or tested. Developing plug-in and scripting support will be very similar to, and intertwined with, developing the data model and will also include sharing and discussing early concepts, sketches, and drafts.

Each day, a **daily scrum** (also called a “stand-up”) will be held. This is a brief meeting where all active team members check in and say what they did yesterday on the project, what they will do today, and whether or not they see a new problem that may become an obstacle to the team’s progress. Remote team members will join via video conference. To keep the meeting brief, there is no discussion. After the meeting, the scrum master addresses any new problems by finding a quick solution or by finding someone who will work on a solution. Any developers blocked from progress on a task will consult the current sprint list to find a new task to work on while the solution is found.

The **backlog** is the set of project goals and the things that need to be completed to achieve these goals. This is managed as a set of lists. The precise set of lists used varies from one project to another and can vary over the course of a project as the team finds they want to split one list into two or rename a list to clarify how it is used. Some teams, where all members are in the same office, maintain these lists using post-it notes on a whiteboard. For this project, we will use the **Trello** web-based list management tool. An online tool has several advantages including supporting remote team members; allowing the client to view the lists; maintaining an automated history of each item, providing automated addition of an item to a list when a new issue is added to GitHub; and providing unlimited space to add comments, checklists, links, and attachments to list items. Trello is used by many programming teams to support agile Scrum project management. The screenshot of Trello (Figure 6-2) shows our management of the proposal writing effort, including developing some proof-of-concept code.

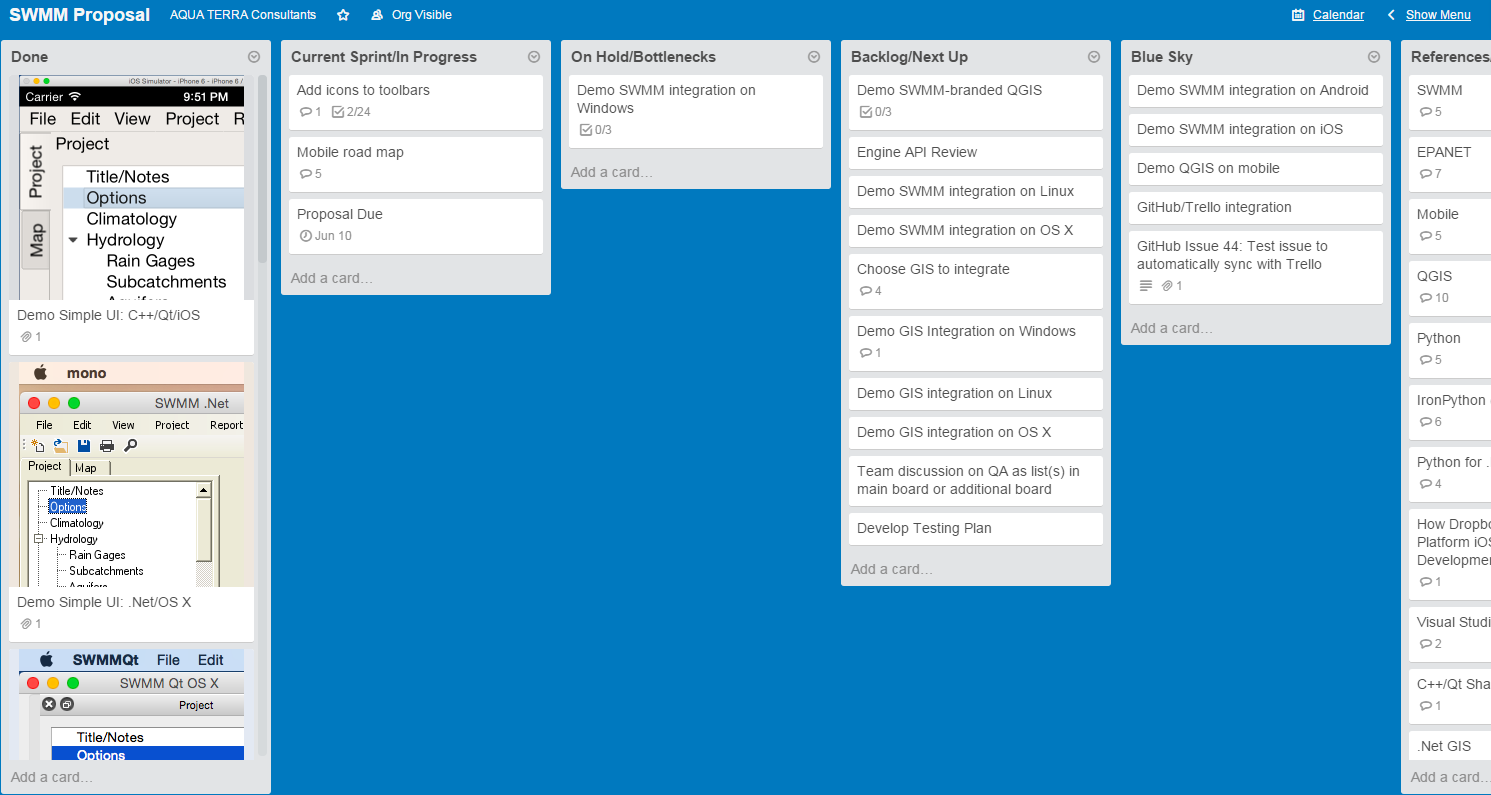


Figure 6‑2. Sample Screen Shot of the Trello Management Tool.

The following set of lists are illustrated:

* **Done:** the work items that are completely finished, tested, and ready to go. Having this list first is good for always seeing a quick peek of what was finished recently. In the screen shot, one can see that we recently finished demonstrating a simple subset of the SWMM interface running on different operating systems and tool chains. Scrolling down that list would show items previously completed.
* **Current Sprint/In Progress:** items that we have agreed as a team to set as goals for the current sprint. In some projects, this is split into two lists with one called Sprint Backlog, where all items for the sprint start, and the other called In Progress, where items are placed when work has actually started. In addition to the option of separating lists, different status of items can also be shown by adding color, icons, or developer avatar icons to items.
* **On Hold/Bottlenecks:** work items that have run into a problem and cannot currently proceed. The reason is noted as a comment inside the item.
* **Backlog/Next Up:** items that are not planned for the current sprint but are available for future sprints. This is a very important list for the product owner to prioritize appropriately because it is the items near the top of this list that are most likely to be considered for the upcoming sprint and, if the current sprint runs out of items, these may even be started early. Near the end of this list in the screen shot is the item “Team discussion on QA as list(s) in main board or as additional board.” This refers to the fact that there could be one or more QA lists between Done and In Progress, depending on the QA workflow. If QA becomes sufficiently complicated to warrant three or more lists, it may be worth creating a separate board just for QA for this project.
* **Blue Sky:** items are all the ideas about what might ever be accomplished for the project. This list is initially populated from the contract. Items that are required by the contract should be given higher priority (position on the list) by the product owner. Other team members may add items at any time (adding to the end of the list since only the product owner assigns priority.) This list is a convenient “parking place” for ideas that may get completed as time and priority allow or may be incorporated into proposals for what could be completed in the next phase.
* **References/Resources:** helpful web page links, notes, and documents relevant to the project. For example, if someone mentions a useful website in a meeting, the link should also be added for easy retrieval at a later date.

## Communication and Meetings

Successful completion of this project will require close coordination with the TOCOR and EPA staff. A variety of communication methods are available for communication project status and receiving client feedback. All of these methods will be explored to determine which are most satisfactory to the EPA.

The most detailed communication method will involve sharing project management tools, such as Trello boards and the project’s GitHub page. These will allow the EPA to track daily progress as much or as little as they wish.

Notable project milestones will be communicated to the EPA via web meetings. The completed sprint may be communicated through review of GitHub development reports where component status (e.g., completion/testing status, version, and date) is displayed. A completed MTP will involve a more thorough progress report where features included in the MTP will be demonstrated and reviewed. A Python notebook, such as iPython or Jupyter, may be a useful tool for demonstrating deliverable features. Other potential delivery methods may be explored and coordinated with the EPA to establish the most satisfactory approach for product delivery.

While the digital age allows for a variety of communication methods, great value is seen in personal, face-to-face meetings. As part of our proposed management plan, a set of four on-site meetings are recommended in the EPA offices, each lasting 2 days. Two core, project team members will attend each meeting. Two meetings are proposed for the first year and one meeting is proposed for each option year.

## Project Risk Management

With the extensive experience our project team has in the areas of software development, support for multiple and cross-language development, object-oriented design, and simulation of complex environ­mental systems, the risks associated with this project are minimal. We created and maintain a widely used platform for modeling support in the EPA’s BASINS, which includes a standalone desktop application with a plug-in architecture and integrated scripting support.

While the core project team does not have extensive experience with Python and PyQt, we have assembled a project team with a very broad range of expertise, including Python and PyQt applications. Since award of this project’s contract, AQUA TERRA has merged with RESPEC, which provides even broader expertise in software development, particularly with Python. The project team will use additional expertise from within our respective companies as needed.

While the project team does not have extensive expertise in EPANET, the current EPANET UI and user manual makes the functionality of this program intuitive to understand. CHI is an accepted SWMM expert and has experience with EPANET as well. Project risks are also minimized because EPANET operates similarly to SWMM and file input structure is clearly documented.

Agile programming methods will also help reduce project risk. The frequent iterative development cycles will ensure the EPA’s early and frequent review of developed products. Early versions of functioning SWMM and EPANET UI will allow the EPA to provide useful feedback to the project team well in advance of final deliverables.

## Deliverables and Schedule

The anticipated schedule for completing the task efforts is summarized in Table 6-1.

Table 6‑1. Deliverable Schedule

|  |  |
| --- | --- |
| **Task** | **Completion Date** |
| Quality Assurance Project Plan (QAPP) | November 2, 2015 |
| Application Features Requirements Document (AFRD) | November 17, 2015 |
| Software Application Architectural Design Memo | January 15, 2016 |
| Proposed Schedule of Minimum Testable Products (MTPs) | January 15, 2016 |
| Proposed Schedule of Minimum Testable Products (MTPs) | January 15, 2016 |
| Management/Delivery of MTPs | Ongoing |
| Computational Engine Support | Ongoing |
| Testing of Final MTP | Ongoing, after last MTP |
| Final Products | 30 days after testing |

# 0 References

**Council for Regulatory Environmental Modeling, 2009.** *Guidance on the Development, Evaluation, and Application of Environmental Models*, EPA/100/K-09/003, prepared by the Council for Regulatory Environmental Modeling, U.S. Environmental Protection Agency, Washington, DC.

**U.S. Environmental Protection Agency, 2003.** *Information Quality Guidelines: Pre-Dissemination Review Guidance and Checklists*, version 2.2, prepared by the U.S. Environmental Protection Agency, Office of Water, Washington, DC.

**U.S. Environmental Protection Agency, 2002.** Gu*idance for Quality Assurance Project Plans for Modeling*, EPA QA/G-5M, prepared by the U.S. Environmental Protection Agency, Office of Environ­mental Information, Washington, DC.

**U.S. Environmental Protection Agency, 1999.** *Documentation for the FRAMES-HWIR Technology Software System, Volume 9: Software Development and Testing Strategies*, prepared by Pacific Northwest National Laboratory, Richland, Washington.

**Van Rossum, G., 2001.** “PEP 0008 -- Style Guide for Python Code," *python.org*, created July 5, 2001, posted August 1, 2013, retrieved October 30, 2015, from *https://www.python.org/dev/peps/pep-0008/*