ActivitySim Strategic Development and Contribution Plan

The purpose of this document is to describe the ActivitySim interagency collaboration and software platform strategic development and contribution plan. This plan provides guidance on coordination and management of the effort as the ActivitySim activity-based travel demand modeling community continues to grow. This plan is focused on strategic topics and therefore details such as contracting mechanisms and guidance for new members is available in other online resources at www.activitysim.org.

This plan discusses the following topics:

- The principles and tenets that guide ActivitySim development
- The methods for managing contributions from others
- The methods for maintaining support for multiple model implementations
- The methods for managing issues and requests
- The methods for managing and prioritizing future work items

Principles and tenets that guide ActivitySim development

The original vision[1] for ActivitySim was to develop a common transportation modeling platform. A consolidated platform will reduce the overall costs of maintenance and development of new model components. Each MPO Partner will have the opportunity to benefit from enhancements collectively identified by Agency Partners, and a common platform should expand the modeling knowledge base. The Agency Partners will further benefit from bug fixes, model enhancements, and performance improvements identified and completed by their fellow colleagues.[2] From this original mandate, we can identify several principles to guide ActivitySim development that are listed in the table below and discussed in the following section.

[1] https://github.com/ActivitySim/activitysim/wiki/presentations/AMPORF.pptx
[2] Association of Metropolitan Planning Organizations Research Foundation Request for Proposals for Consolidated Travel Model Software Platform Development and Enhancement 2015.

Principles and tenets

Principle	Brief Description
Collaborative	One open common platform / code base that is shared by all users
Cost effective	Reduced development and maintenance costs and economies of scale through pooled funding
Practical	Easy for agencies and modelers of different skill levels to use to produce reasonable and reliable estimates and forecasts
Extensible	Can be customized and extended for new features and region specific needs
Performant	Makes efficient use of computing resources, including memory, storage, and processors

Collaborative

The need for ActivitySim grew from the typical consulting software model of starting a consulting project by making a copy of an existing code base and then customizing it for agency needs in a user-friendly manner. While this reduces startup costs, it often causes more problems in the long run since the software platform appears to be open and common across projects but is actually diverging over time as improvements for one user are difficult to share with others. The challenge therefore becomes to truly share one common platform (i.e. one code base) across all users. This means establishing an inclusive way of working together, along with a technology / solution, that operates across projects, agencies, contracts, and schedules for the benefit of everyone. In addition to the technology, an advanced state of the practice set of example models that can be customized for user needs is necessary to demonstrate and test the software. This project also has the benefit of providing best practice example models, grounded in proven data driven methods, that are informed by the cumulative efforts of the agency partners. ActivitySim is therefore an agreement by

agency partners to develop and maintain a shared common platform for activity-based travel demand modeling that can be extended and customized based on partner needs.

Cost Effective

It is expensive to develop and maintain high quality complex multithreaded microsimulation activity-based travel demand models that evolve with agency planning needs. The founding members of the ActivitySim project spent millions of dollars developing pioneering custom software tools. In addition, much of the work to develop these custom software tools was done under consulting projects, which have a clearly defined beginning and end, which is inconsistent with a software tool in everyday use, and the need for maintenance and support. As the costs, both in time and money, to develop and maintain these evolving tools remained significant, the need for a more financially sustainable business model for an activity-based travel modeling platform became evident. By pooling funds, agency partners are likely to benefit from economies of scale and a higher rate of return on investment. In addition, over time it is expected that agency monetary contributions to develop and maintain the platform will go down as there are more partners participating, there is less work to do as the platform matures, and there are code contributions not directly funded by the consortium.

Practical

The goal of the ActivitySim project is to develop an activity-based travel demand modeling platform for practice. A practical software platform is used to inform local, regional, and state transportation planning analysis and is easy for agencies and modelers of different skill levels to use to produce reasonable forecasts. Several agencies, including many of the ActivitySim agency partners, have been using activity-based models for years in practice. At a minimum, ActivitySim must continue to serve these needs. ActivitySim must also be user friendly. Activity-based travel demand models are intricate tools with several user inputs, lots of input settings/parameters, hundreds of data processing / mathematical expressions, plus lots of big input files, and are therefore ripe for accidental misuse and error. To help alleviate the risk of accidental misuse and error, ActivitySim was built on top of the mature and popular open source data science Python libraries pandas[3] and numpy[4]. These tools have large user communities and a wealth of online resources to support

[3] https://pandas.pydata.org/
[4] https://numpy.org/

easier use. ActivitySim also took the approach of separating code from model configuration and utility expression specification in order to separate software engineering from travel modeling. Easy-to-use software is also well documented, including any assumptions around inputs, outputs, and methods.

Extensible

An extensible system is one that allows for the addition of new capabilities and functionality. The transportation and travel modeling industry is full of innovation - new mobility services, more behaviorally accurate models, more precise treatments of network impedances by mode and user, etc., and the software tools to support exploration of these innovations must keep up to satisfy stakeholder needs. At the same time, developing and maintaining a common platform that is both used in practice as well as capable of incorporating innovations is challenging. ActivitySim started with the idea that the MTC (San Francisco Bay Area MPO) CT-RAMP activity-based travel demand model used for several transportation planning applications could be re-implemented with modern open source data science libraries, and then extended for use by other agencies. Previous activity-based modeling software tools often struggled to support extensible feature specificity, typically due to differences in both model design and software design. Extensible systems share a core set of functionality used across implementations, while also providing components and interfaces (or APIs) for customization. As the ActivitySim user community continues to grow, and diversify, the challenges of maintaining an extensible system increase. Managing contributions (from others) is especially important for extensibility.

Performant

Reasonable model runtime performance and resource utilization is essential for producing timely analysis. It is common for activity-based travel models to be run hundreds of times and so efficient use of computing hardware, including memory, storage, and processors, is a requirement. At the same time, the system must be reliable, reproducible, transparent, archivable, and dependable so the user has confidence in application. Finding the right balance of explanatory power versus stability versus runtime is one of the major challenges in travel modeling, and it is true for the ActivitySim project as well. The design of ActivitySim, both today and tomorrow, will always need to balance its ability to reasonably and reliably answer planning questions with its need to do so in a performant manner.

Managing contributions from others

The ActivitySim project welcomes contributions by members and non-members. It is anticipated, and encouraged, that the community will contribute to the collaborative platform. As the community is expected to grow, so too will the need and resources to manage this sophisticated software platform.

Purpose

The purpose of this section of the plan is to describe the methods for managing contributions from others. Before discussing the approach, we review historic practices since it helps us better understand how to plan for the future. A summary of the key aspects of the approach is provided at the end of the section.

Background

For the first few years of the project, the development roadmap centered on re-writing MTC travel model one, updating it to use modern Python data science tools, and adding important user functionality. In 2020, some significant new features were added, including estimation integration inspired by DaySim and support for multiple zone systems and transit virtual path building inspired by some newer CT-RAMP models. Looking forward, the development roadmap is less clear, and so contributions that align with the development roadmap are also less clear. Thus, a clear development roadmap is desired.

Probably the single most important characteristic of the ActivitySim project is the goal to establish a collaborative platform for activity-based travel modeling that the community will improve over time. In order for this principle to succeed, a robust framework for managing contributions from others is required.

As discussed earlier, previous open source activity-based modeling software projects, such as CT-RAMP and DaySim, were not built with contributions from others in mind. In both cases, the code base was typically copied from one region to another with essentially no mechanism (or link) between the versions for incorporating contributions from others. More recently, most versions of DaySim have been unified under a new core version of DaySim maintained on GitHub, whereas the versions of CT-RAMP have continued to be standalone.

With the creation of the ActivitySim project, and the membership of agency partners including CT-RAMP and DaySim-based model systems, most agency resources geared toward improving software collaboration and improvements have been oriented toward the new, more capable, and openly engineered and maintained ActivitySim platform.

As of Fall 2020, the development of ActivitySim has largely been done by one contributing team at a time. While there have been a few small contributions from others, the majority of work has been done by the team responsible for also managing the repository. As a result, it could be argued that ActivitySim is yet to significantly exercise the principle of a collaborative and extensible platform.

However, ActivitySim has some solid existing infrastructure for managing contributions from others, including using GitHub Flow[5] for working together in the repository and the existing contribution review criteria[6], which have both been adopted by other open source efforts. A specific criteria to highlight is the requirement that contributions implement good methods (i.e. they are consistent with good practices in travel modeling). This is especially important when reviewing contributions from others as it reinforces the practical principle. These processes, plus others, serve as the foundation for the approach.

Approach

The approach for managing contributions from others includes a development roadmap, repository manager, a collaborative site for tracking third-party contributions, a review process and agreed upon workflow, and an inclusive contribution community.

Development Roadmap

A formal development roadmap needs to be added to the project's management resources and be posted online for potential contributors to review. At a minimum, this will identify likely future work items, new modeling and software components being considered, desired timeline for inclusion, approximate level of effort, and any significant known issues. In conjunction with establishing a roadmap is to make better use of versioning/releases so designs about "version 2" or "version 3" can more easily take shape through the collaborative development roadmap. With a development roadmap

[5] https://guides.github.com/introduction/flow/
[6] https://github.com/ActivitySim/activitysim/wiki/Contribution-Review

in place, review and communication around contributions from others will be more straightforward and less reactive.

Repository Manager

Implementation of the process management starts by appointing a repository manager. This role on the project is responsible for ensuring the project software repository and test system is always in good working order. This role is responsible for the following items and, this role, ultimately reports to the project management committee for final approval of contributions.

- Ensuring the Continuous Integration (CI) test system is always working
- Ensuring proper use of GitHub Flow, as described below
- Ensuring coding according to pycodestyle, the tool used to check code against the pycodestyle style conventions
- Ensuring good documentation, including all relevant resources such as user guides, Jupyter notebooks, wikis, etc.
- Ensuring good test coverage
- Ensuring good modeling practices
- Ensuring proper use of GitHub issues for issues, feature requests, questions and support
- Ensuring proper versioning and release procedures, as explained below
- Coordination with the project management committee for contribution review and approval as needed
- Ensuring the third-party contributions site is up-to-date, as described next

Third-Party Contributions

Interest in contributing to ActivitySim continues to grow. To assist the project management committee and ActivitySim users and developers with tracking and publicizing potential third-party contributions, an online site will be established. The site would be managed through the GitHub project wiki and include topics such as:

- Planned contribution description:
    - Methods
    - Potential new dependencies
    - Alignment with development roadmap
- Team/individual working on the effort
- Expected timeline

- Potential / requested areas of assistance for contribution (coding assistance, documentation assistance, example development assistance, etc.)
- Validation and discussion of the methods

GitHub Flow

The contribution process should continue to use GitHub Flow. The key points to GitHub workflow for ActivitySim are:

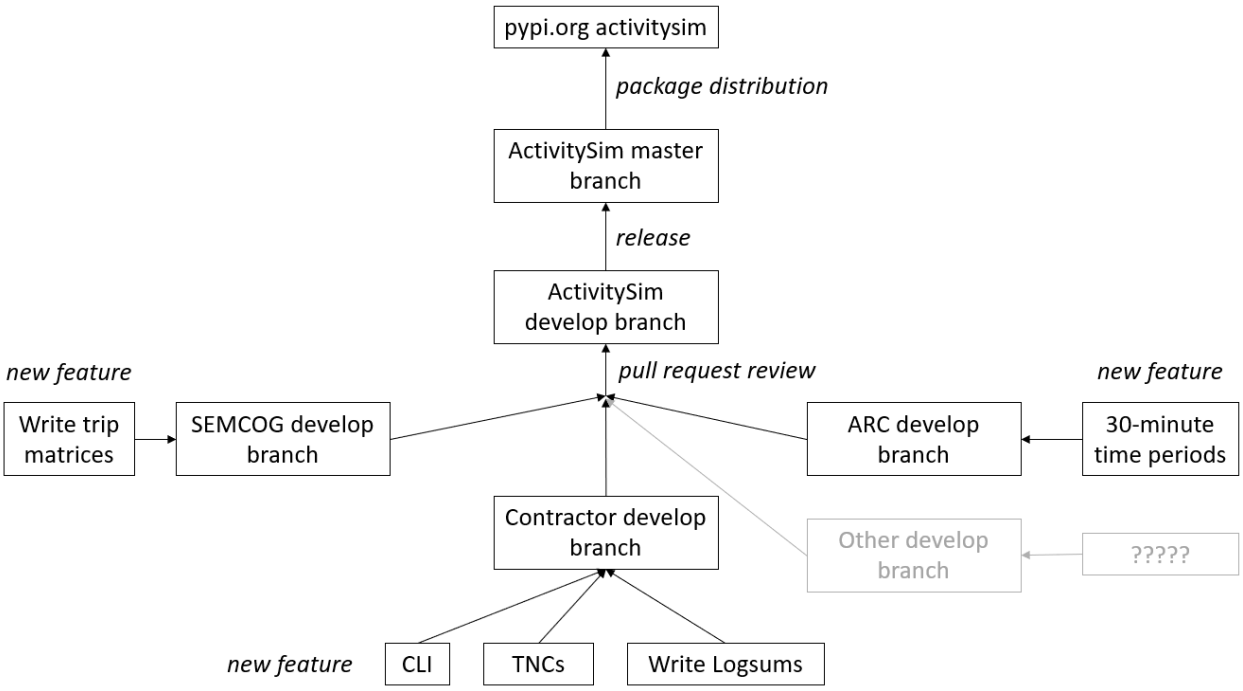
- The master branch contains the latest working/release version of the ActivitySim resources. The master branch is protected and therefore can only be written to by the CI system.
- Work is done in an issue/feature branch (or a fork) and then pushed to a new branch.
- The test system automatically runs the tests for the examples on the new branch. In some cases, test targets need to be updated to match the new results produced by the code since these are now the correct results.
- If the tests pass, then a manual pull request can be approved to merge into master
- The repository manager handles the pull request according to the contribution review criteria and makes sure that related resources such as the wiki, documentation, issues, etc. are updated.
- The repository manager coordinates any review concerns with the project management committee and makes revisions as necessary.
- The responsibility for fixing errors or bugs identified in the code review is the responsibility of the contributing author. It is not the responsibility of the reviewer.
- Every time a merge is made to master, the version is incremented and a new package posted to pypi.org. Versioning follows the major.minor format discussed next.

ActivitySim uses the MAJOR.MINOR versioning convention. MAJOR designates a major revision number for the software, like 2 or 3 for Python. Usually, raising a major revision number means adding several new features, breaking backward compatibility or significantly changing the APIs / interfaces / contracts. MINOR usually designates moderate changes to the software like bug fixes or minor improvements. Most of the time, users can upgrade to a new minor release with no risks to their software. Because major releases have significant ramifications, the project management committee decides when to identify, develop, and release a major version.

The figure below helps illustrate the process of working together across and within the ActivitySim repository. The key parts of the figure are:

- In the center of the diagram is the ActivitySim organization account master repository develop branch. This is the central clearinghouse for all ActivitySim related work.
- Eventual contributions are developed in a forked develop branch, such as in the SEMCOG or ARC organization account fork of the ActivitySim organization account repository. A new feature is developed in the SEMCOG fork and then a pull request is issued from the SEMCOG fork to the ActivitySim master repository. This pull request includes updated documentation and tests to describe and exercise the new software as well. This pull request is not pulled until the review process below is complete.
- At the same time, in a separate fork, such as the ARC organization account fork, developers may be working on a new feature. This effort could also create a pull request from its repository to the master ActivitySim repository.
- Once the repository manager reviews the pull requests according to the contribution review criteria, and the contributors meet all the criteria, then the contributions (including code, documentation, and example tests) are pulled, or incorporated, into the master repository develop branch.
- Either at the same time, or at a later date once a bundle of improvements have been collected and approved by the project management committee for release, a release of ActivitySim is made by pushing the revisions from develop to master, building the online user guide, and then posting a package release on pypi.org. This process is the same for major or minor releases, with major releases requiring additional review by the agency partners.

Example Contribution Management Workflow



The benefits of this approach to managing contributions from others is it provides a linked framework for multiple versions of the software along with mechanisms for intelligently merging but first reviewing and improving contributions. It is also entirely online and integrated with GitHub issues and the test system, which makes the overall process transparent and more collaborative.

Contributor Recognition and Release

Each contributor should be recognised and feel a productive part of the community, and it is a goal of the project to encourage diverse industry participation. Key to this is the recognition of contributions from individuals in a manner that also recognises the community effort that made it all possible. Contributions of actual code, documentation, and example tests are supported by design discussion, oversight, testing, documentation, bug fixes and much more. It is therefore impossible to credit individual contributors.

However, contributions are tagged to the contributing GitHub account, which makes it possible, at least partially, to identify contributors. In addition, contributors are

encouraged to create issues and then tag issues in their commits in order to better manage the software and to leave a trail of contribution.

ActivitySim is an open source project that is distributed at no charge to the public. All contributions to ActivitySim must adhere to the license[7], and contributors must acknowledge in the contribution review criteria that they have an official release of ownership from the funding agency if applicable. This ensures that the ActivitySim project has the right to freely distribute the software.

Managing Contributions from Others Summary Table

Approach	**Brief Description**
Maintain a development roadmap	● Establish a development roadmap to help coordinate potential contributions from others
Establish a repository manager role	● The repository manager is responsible for ensuring the repository and test system is always in good working order ● The repository manager is responsible for coordinating review with the project management committee when needed
Establish site for third-party contributions	● Maintain a resource for the develop community to track / publicize on-going and future contribution plans
Continue to use GitHub Flow	● Use the industry standard framework for collaborative open source software development
Adhere to the Contribution Review Criteria	● Follow, and revise when needed, the project's contribution review criteria and process

[7] https://github.com/ActivitySim/activitysim/blob/master/LICENSE.txt

Version releases	● Follow MAJOR.MINOR convention with the project management committee deciding when to identify, develop and release major new versions
Maintain supportive contribution environment	● Ensure contributors feel welcome, software is properly licensed, and contributions are legally released to the project

Maintaining support for multiple model implementations

The ActivitySim consortium is a growing community of diverse model implementations. Being able to successfully maintain a shared common modeling platform across implementations will be key to project success.

Purpose

The purpose of this section of the plan is to describe the methods for maintaining support for multiple model implementations with one shared common platform. Before discussing the approach, we review the challenges and historic approach since it helps us better understand how to plan for the future. A summary of the key aspects of the approach is provided at the end of the section.

Background

Historically, activity-based modeling software for a new implementation, often a new region for example, has been implemented in one of two approaches:

- Implement a new code base that was independent of other code bases
- Copy an existing code base and modify it as needed for the new implementation, including primarily changing configuration options but also code as needed

There are some gradations among these approaches, but the basic idea was the same - to start anew for each implementation with no clear way to contribute and test improvements from other users. While the starting code base had some configuration options, it was typically incomplete for the next user and so some code revisions were required due to the challenges described below. In terms of testing, this was typically

done through actual use (i.e. there was no test system, just running the actual model, making sure everything works as expected, and revising as needed). As explained later, it was not until this project that a serious attempt to develop a common platform, including contribution framework and test harness, for activity-based modeling was attempted.

Model implementations across regions, states, user types, projects, etc. can be significantly different, thereby making maintenance of the entire suite of models challenging. In order to better understand approaches to maintaining multiple models, a review of key differences in implementations follows.

- Differences in input data - land use attributes, synthetic population attributes, employment types, school enrollment types, spatial resolutions (TAZs, microzones, parcels, transit stop areas (or access points)), network level-of-service (skim) attributes and time periods, global value-of-time settings versus submodel specific value-of-time settings for example, etc.
- Differences in alternatives - modes, activity types, etc.
- Differences in segmentation - person types, work status, school status, etc.
- Differences in submodels - telecommute model, CBD parking location model, time-of-day probability lookup tables versus choice models, etc.
- Differences in submodel dependency, sequencing, and interaction - explicit modeling of joint travel, destination choice before mode choice or vice versa, etc.
- Differences in core components - traditional choice models such as MNL and NL versus new models such as MDCEV, one zone model system versus two zone model system, etc.

Some of these challenges have historically been straightforward to accommodate, whereas others have been difficult. For example, differences in land use attributes, synthetic population attributes, employment types, and network skim attributes are often easier to accommodate because the structure of the data remains the same - there is just replacement or additional data along existing dimensions. Differences that require revisions to data structures, such as new submodels or interactions between submodels, for example tracking of joint travel (where multiple persons make the same tour or trip) throughout the model stream, are more challenging since the code that consumes these revised data must typically also be revised.

Approach

The method for maintaining support for multiple model implementations that share a common modeling platform consists of the following key ideas:

- Exposing settings for differences in input data
- Exposing configurable options for differences in alternatives, utility expressions, and segmentation
- Exposing submodel contracts for differences in submodels and submodel interdependencies and interactions
- Exposing core component interfaces for differences in underlying components
- Sufficient test coverage of the functionality in the multiple model implementations

What follows is a detailed discussion of each idea, followed by an example to illustrate implementation of the approach.

Exposing settings for differences in input data

The first level of user configuration starts with differences in inputs. This first level of differences between model implementations is handled through user defined input tables (such as zonal land use data, synthetic persons, synthetic households, and skims), global and submodel specific parameters (such as skim time periods and the maximum walk to transit distance for mode choice), and submodel expression and coefficient files (i.e. model alternative utilities). Each ActivitySim model setup defines global settings available to each submodel, as well as submodel specific settings. For example, if a user has a different set of employment types, then they would need to provide their land use data file, as well as, revised downstream submodel settings, such as expressions files for accessibility and destination choice that make use of the new data. These settings are exposed via YAML files for the user to edit and ActivitySim accepts inputs in open data formats such as CSV and OMX.

Exposing configurable options for differences in alternatives, utility expressions, and segmentation

A more complex, but still relatively straightforward difference between implementations are more substantive configurations such as new/different modes of travel, different utility expressions, and market segmentation such as activity type, person type, and school type, which may be used to apply a specific school submodel specification to a specific type of person. Unlike the first set of differences, these differences start to get

more at differences in model design. Similar to the approach for handling differences in input data, each ActivitySim model setup defines submodel specific settings for differences in alternatives, utility expressions, and segmentation. For example, if a user has a different set of modes, such as walk to premium transit and walk to local transit, instead of just walk to transit, then they would need to configure all relevant submodels and their configuration files - settings, expressions, and coefficients. In addition, if a model solves a different set of expressions by market segment, for example, K-12 versus University location choice, then the user configures the segmentation via the same set of files. These settings are exposed via YAML and CSV files for the user to edit. Experience deploying ActivitySim for SEMCOG and ARC resulted in a few small pull requests for improvements in this respect. Nevertheless, as additional ActivitySim deployments are set up, it is important to continue to expose the alternatives, utility expressions, and segmentation via the submodel config files.

Exposing submodel contracts for differences in submodels and submodel dependencies, sequencing and interactions

The ActivitySim software package has two primary subpackages - models and core. The models subpackage contains specific modeling steps such as auto ownership, tour frequency, and trip mode choice. The core subpackage contains framework components such as logit models, expression handling, and multiprocessing that are used by the submodels to implement a complete model design.

As noted above, the current platform allows for customization of modes, utilities/expressions, etc. but not for adding/revising/modifying submodels without writing or updating the Python code. If a user wants to implement a different version of a submodel, such as a different version of the trip scheduling model, then they would need to create a new trip scheduling submodel, named differently, and revise dependent modules, core functionality such as possibly the person time window availability code, and update the test examples to test the new submodel. This process is ad hoc and depends on the specific submodel since ActivitySim maintains no formal definition for a submodel and its relationship with other components, i.e. its contract. A submodel contract that defines expected inputs, outputs, methods, and dependencies would make accommodation of differences in model implementation easier.

Closely related to adding new submodels is a more significant type of revision: resequencing submodels. Resequencing submodels, or more generally revising the overall model system design, typically means significant revisions to the dependencies / interactions between submodels, between core components, and between submodels

and core components. In the case of resequencing submodels or developing new overall model designs, it is sometimes better to extend a submodel or core component to support additional use cases, where as other times it is better to create a new submodel or core component that works only for the new use case (at the time of creation). The preferred approach depends primarily on how similar the revisions are to the existing software.

A third case might be to just implement one submodel, such as a new and improved version of trip mode choice. In this case, instead of running a complete set of travel submodels from beginning to end, ActivitySim runs just the one submodel and therefore requires a clear contract for that submodel alone with respect to inputs and outputs.

In any case, the submodels and core components require certain inputs, implement certain methods, and produce certain outputs, and the framework should provide the developer with a documented and easy way to provide them - e.g. an contract. ActivitySim should continue to better define and implement a more formal submodel contract that delineates how inputs, outputs, methods, dependencies, settings, etc. are handled and how this contract is understood and registered with the platform so the exercise of revising or adding new submodels and/or core components is more straightforward. This should help support implementation of new features as the user community grows.

Exposing core component contracts for differences in underlying components

Differences in core components are probably the most difficult differences to maintain across multiple implementations of ActivitySim. In 2020, the ActivitySim platform began the transition from essentially supporting one model design with some customization functionality to supporting multiple model designs by adding support for models with one, two, or three zone systems. In addition, the addition of model estimation functionality required significant revisions to the platform plus the creation of a new estimation mode test example. Both of these development efforts led to significant revisions to the relationships (i.e. contracts) between core components such as expression management, tracing, skims (or network level-of-skims information in general), and multiprocessing, and to a lesser extent, the submodels.

Since new features are expected to be added through a collaborative contribution model, good documentation and clarity around core component technology is essential for the development community. For example, newer choice models such as MDCEV

select multiple alternatives instead of just a single alternative like MNL and NL and therefore require a different contract with their calling software components.

Like submodels, continuing to better define and provide a more formal framework for evolving contracts for core component technologies will make it easier to add new features. This framework should be documented online so as to be accessible to the open source community.

Sufficient test coverage of the functionality in the multiple model implementations

The test system provides a test bed of examples that exercise the platform under diverse uses to help ensure software reliability/stability and futureproofing, and by providing expected / known answers to help verify that changes made to the software are actually the changes that were expected. The test system also sets up a clean installation of ActivitySim to check dependencies, checks the source for style guide compatibility, builds the user documentation based on comments embedded in the code, and deploys the online user's guide. The ActivitySim test system, which tests not only component functionality and feature behavior, but also complete model example runs from start to finish, is central to addressing the challenges described earlier.

A good test system has high code coverage (i.e. the percent of the code that is exercised by the tests). Often the necessary data to test a new feature is only available for the new implementation (e.g. for a new region as opposed to one already in the test system). As a result, the developer of the new feature essentially has two options for creating test data to go along with software contributions:

- Create new input and output test data for existing implementations (e.g. regions / model designs) already in the test system
- Add (a subset of) the new implementation and its test data to the test system

Both capabilities will need to be supported to ensure adequate test coverage for maintaining support for multiple model implementations. Existing implementations serve as the core examples for testing and developers are encouraged to update these when possible. Alternatively, when the design and features of the new implementation are different enough from the existing examples, then (a subset of) the new implementation would be added to the test system. Like some of the existing test examples, when adding test coverage for the new implementation, only a subset of households and/or zones can be added so test runtimes are manageable. Adding (a subset of) the new implementation has the additional benefit of offering some additional

assurances that future versions of ActivitySim will more easily work for the model user since (a subset of) their model would have already been tested.

For agencies interested in having their complete ActivitySim model periodically tested against software updates, an optional agency specific test system could be established. This would be similar to the DaySim test system, where agencies periodically share a complete ActivitySim model setup via GitHub, and this test system checks out the latest version of ActivitySim and runs each agency model to completion or to identify issues. Because of the large data files and potential runtimes (even with sampling), this test system may be run nightly or weekly instead of with each commit. The benefit of this system for agencies is additional future proofing that updates to ActivitySim will more likely work when used in practice. An additional benefit for ActivitySim developers is the ability to get additional test coverage when developing new features.

Maintaining Support for Multiple Model Implementations Summary Table

Approach	**Brief Description**
Exposing settings for differences in input data	● Provide, and enhance when needed, user options to configure differences in inputs via settings files
Exposing configurable options for differences in alternatives, utility expressions, and segmentation	● Provide, and enhance when needed, user options to configure differences in inputs via settings files
Exposing submodel contracts for differences in submodels and submodel dependencies, sequencing and interactions	● Add submodels to the models subpackage ● Develop a formal submodel contract ● Publish the submodel contract ● Refactor existing examples to implement revisions ● Require new features to implement contract

Exposing core component contracts for differences in core underlying components	● Revise core subpackage to work with new and existing models in the models subpackage ● Develop a formal contract for core components ● Publish the core components contracts ● Refactor existing examples to implement revisions ● Require new features to implement contracts
Sufficient test coverage of the functionality in the multiple model implementations	● Extend test examples with appropriate test input and output data with each revision ● Add (a subset of) new implementation test data to test new features if needed

Example: Transit Capacity Constraint, Crowding & Reliability

In order to illustrate how the ActivitySim platform can support additional or enhanced features, we consider an example where a region wants to incorporate transit capacity constraint, crowding and reliability (TCCR) into their ActivitySim implementation. In brief, incorporation of TCCR means revising the travel demand and supply models to be sensitive to transit vehicle and parking lot capacity and service reliability. Assuming the network model is revised to produce additional skims to reflect network crowding and reliability measures, and the model system is a three zone model system, then the following set of revisions to ActivitySim may be in order:

- Differences in input data
    - New skim inputs to reflect network crowding and reliability measures
    - Revised transit virtual path building, accessibility, and mode choice utility specifications to include these new skim measures
    - Revised coefficient files that parameterize these new measures
- Differences in alternatives
    - No changes to the alternatives are expected
- Differences in segmentation
    - No changes to the market segmentation are expected
- Differences in submodels
    - Revised mode choice submodels that compare TAP demand to supply (capacity) by iterating until an acceptable level of convergence is met
- Differences in submodel dependencies, sequencing and interactions

    - New process for iteratively running mode choice and possibly other component models such as accessibility to select TAP pairs based on TCCR within a single global iteration of the demand model. Additional iterative procedures for incorporation of TCCR within the network model would also be developed.
- Differences in core components
    - Revised transit virtual path builder to support TAP demand to capacity calculations and iteration

The table below summarises the revisions to ActivitySim to support transit constraint, crowding and reliability.

Transit Capacity Constraint, Crowding, and Reliability Summary Table

Approach	**Brief Description**
Exposing settings for differences in input data	● Additional skims, utility expressions, and coefficients/parameters exposed via YAML, CSV, and OMX files
Exposing configurable options for differences in alternatives, utility expressions, and segmentation	● Additional skims, utility expressions, and coefficients/parameters exposed via YAML, CSV, and OMX files
Exposing submodel contracts for differences in submodels and submodel dependencies, sequencing and interactions	● Updates to the accessibility models to optionally support TCCR ● Updates to the mode choice models to optionally support TCCR

Exposing core component contracts for differences in core underlying components	• Updates to the transit virtual path builder to optionally support TCCR
Sufficient test coverage of the functionality in the multiple model implementations	• Extend test examples with appropriate test input and output data to test TCCR. Alternatively, add (a subset of) new implementation test data to test TCCR.

Managing issues and requests

The needs of the ActivitySim platform continue to grow as the user base grows. This section describes the approach to better manage issues, feature requests, questions, and support as needed.

Purpose

The purpose of this section of the plan is to describe the methods for managing issues, feature requests, questions, and support. Before discussing the approach, we review historic practices since it helps us better understand how to plan for the future. A summary of the key aspects of the approach is provided at the end of the section.

Background

From 2015 to 2020, the ActivitySim project has principally been handling issues, feature requests, questions, and support through a combination of GitHub issues, email support, and consortium project management calls. This has generally worked fine since the platform has largely been under development and without many users, and so the number of user issues, feature requests, questions and support have been limited.

More recently, interest in ActivitySim, and therefore support for ActivitySim, has started to increase. Support for users beyond consortium members has been delivered through a combination of email, GitHub issues, and phone conversations by consultant and agency members. While sufficient to date, this informal approach to supporting the

broad user community is unlikely to scale with the growing demand and thus a more formal approach to managing issues and requests is needed.

In reviewing the approach of other open source efforts, there appears to be broad consensus for using GitHub issues for issues, feature requests, questions, and support. The asynchronous and archival manner in which users operate has tremendous benefits for the community, including many of the principles of this project: common, sustainable, and usable. Several projects create a set of possible issue labels (or tags) that are assigned to issues and then used to filter and/or aggregate for decision support. Example issue labels are:

- Feature request
- Question
- Bug
- Documentation improvement
- Significant issue

Communication around bug tracking and fixes is especially important as users often want to correct these issues in a timely manner. Many of the projects also maintain a developer / user listserv and/or email account(s). This has the advantage of being somewhat private and not openly archived online, which can be especially useful for questions around governance of the project, more general modeling questions, and other topics less fit for GitHub issues.

The commercial transportation modeling software developer approach to issues, feature requests, questions, and support tends to be more one-on-one email or phone support via a maintenance contract. One could argue that the advantages of this approach are that users get their ultimate issue or question more easily and/or quickly resolved and that the conversation is private. The disadvantages of this approach are that it is not common (i.e. open and archived) and it requires dedicated technical support staff which increases cost to support the platform.

Approach

The approach to managing issues, feature requests, questions, and support is to continue to use GitHub issues with labels, to set up an activitysim.org email account when GitHub issues technology is inadequate, and to establish tiered support. Management of the GitHub issues is the responsibility of the project management committee, who may delegate responsibility if desired. Bugs will be specifically called out and communicated via issues and project management resources since they are

especially significant. When GitHub issues technology is inadequate, an email option such as info@activitysim.org or activitysim@ampo.org will be set up. The project management committee will be responsible for the email account, and they may delegate responsibility if desired. This email address could be part of an ActivitySim/AMPO website. Questions and support can be handled through either GitHub issues or the email account.

A tiered support strategy is recommended. For tier 1, consortium members get guaranteed support with membership. Each phase of work will include a fixed amount of resources for managing consortium member issues, feature requests, questions, and support. These resources will become more critical as the platform is used by agency partners and others for planning studies with schedule constraints for bug fixes and technical support. For tier 2, non-members who desire active support (which means more than simply responding to the occasional GitHub issue), can sign-up for an annual support only option, which is a fraction, but not more than 50% of the consortium membership fee and does not include membership in the project management committee. For tier 3, non-members with limited support needs can get time and materials support via a simple invoice or purchase order with a bench consultant of their choice. This arrangement allows for agency partner resources for answering questions and providing user support to be largely limited to agency partners, while also providing a mechanism for non-members to get support. Finally, regardless of affiliation, ActivitySim is an open and common initiative and so all users, developers, agency partners, and others are encouraged to participate in the GitHub conversation.

In preparation for scoping future phases of work, the project management committee reviews feature requests and compiles them according to the process outlined in the managing future work ideas and prioritizing future work section. Because ActivitySim is a practical data driven framework, future features need to be grounded in observed / verifiable results for eventual acceptance and distribution.

Managing Issues and Requests Summary Table

Approach	**Brief Description**
Using GitHub issues	● Use online collaborative software development tools for public issues, requests, questions, and support

	• Explicitly label and communicate bugs and fixes for timely communication with the agency partners / users
Add email access to the project	• Use a shared email address for private and general inquiry type questions such as consortium management
Tiered user support	• Consortium members get guaranteed support • Include a fixed amount of resources for user support in each phase • A non-member annual support only option is also offered • A non-member on demand time and materials invoice / purchase option is also offered • Regardless of affiliation, encourage involvement via GitHub
Incorporate feature requests into scoping of future work	• Review and incorporate feature requests into subsequent work phase scoping exercises

Managing and prioritizing future work items

The ActivitySim consortium is a growing community of travel modelers with diverse needs. Managing ideas about future work and prioritizing improvements to the platform requires thoughtful and coordinated cooperation among members to be successful.

Purpose

The purpose of this section of the plan is to describe the methods for managing ideas about future work, prioritizing that work, and then converting that work into executed work. Before discussing the approach, we review the historic work plan since it helps us better understand how to plan for the future. A summary of the key aspects of the approach is provided at the end of the section.

Background

The ActivitySim project, which is a software project first and foremost, is focused on developing a collaborative software platform for activity-based modeling, as opposed to developing a standalone model system. A software platform, as opposed to a model system, is a customizable framework for implementing multiple diverse model implementations, as opposed to a version of a specific, or one-off, model design. The ActivitySim project believes examples of prototypes of new models are prevalent in the industry and that examples of collaborative, cost effective, practical, extensible, and performant platforms are few.

Historically, ActivitySim's work program has been centrally focused on re-building the existing MTC TM1 mode. TM1 was one of the first activity-based models for a major metropolitan area to be used for several planning studies, including the Regional Transportation Plan, a major function of the MPO. TM1's demand model is an implementation of the Java-based CT-RAMP platform. In early 2019, the ActivitySim project essentially completed the re-write of TM1 into a more common platform software solution implemented in Python. The partners chose to re-factor TM1 rather than design, estimate, implement, and calibrate/validate a new model system since TM1 was proven and highly capable, verification of the new version could be done against the previous version, and the emphasis of the initiative was on software engineering as opposed to model design.

ActivitySim work to date has been done in Phases, each with a discrete set of tasks. Phases are typically six months or twelve months in length and result in a release of the work. With a focus on re-factoring TM1, the historic work plan has been dictated by completing the features of TM1, while at the same time, improving usability along key topics such as configurability/flexibility, documentation, test coverage, data management, data pipelining/restartability, etc. Each phase of work has typically included a focus on additional TM1 model components plus some usability enhancements. In 2019, with the completion of the TM1 re-write, the future work program became less obvious.

The founding members of ActivitySim, MTC, ARC, and SANDAG all shared a CT-RAMP model. However, while the activity-based model design for MTC and ARC is quite similar, the SANDAG model had a significant difference: three zone systems. This three zone system version of CT-RAMP was a separate code base from the single zone system version of CT-RAMP and so sharing of code improvements was quite difficult. In addition, the implementation of all three models was somewhat exploratory (i.e. it

contained several code blocks of thought-to-be-good-ideas that were later no longer needed) and hard coded references to components likely to vary in different implementations (such as time periods, modes, and constants). In 2020, the ActivitySim project began the exercise of developing support for a system that works with either a single zone system, two zone system (see below), or three zone system model.

As interest in ActivitySim developed over the years, two agencies with existing activity-based models implemented in the C#-based DaySim framework joined the effort. These two agencies were SFCTA and PSRC. DaySim's design has many similarities to CT-RAMP but there are some major differences:

- Additional support for two zone system models
- Model re-estimation functionality
- A different approach to modeling time-of-day
- A different approach to accessibilities
- A different approach to consistency across tour and trip mode choice through a shared pathtype model

In 2020, the ActivitySim project implemented support for two zone systems and model re-estimation functionality, and additional reconciliation and implementation of CT-RAMP and DaySim functionality in ActivitySim is likely as the project progresses.

Looking forward, there will always be improvements to the model design and the software platform that are desired, and an approach to manage ideas around future work and prioritize and execute this work is required.

Approach

In order to manage ideas around future work and to prioritize and then execute that work, a development roadmap and an ideas tracking and prioritization process is needed. Based on the principles described earlier, the roadmap and process should be collaborative, affordable, practical, easy-to-use, and sustainable.

Development Roadmap

As described under Managing Contributions from Others, a formal development roadmap is needed for managing and prioritizing future work items. This online collaborative development roadmap will help organize discussions and plans for future

work and will have the benefit of informing non-members and potential members of ActivitySim's plans.

Process

The basic process to manage future work ideas and prioritization, which is inspired by the 2020 Phase 6 scoping[8] exercise, is outlined below. This process may be revised / adapted as needed, but should generally be followed since it is inclusive and comprehensive.

- Describe Potential Work - Agency partners and team members describe desired features for the next phase of work. At this point in the process, all ideas are described regardless of level of effort, data needs, or other potential constraints. This list is informed by the potential third-party contributions site as well. Agency partners and team members comment, discuss, and refine the feature descriptions if needed. These ideas are maintained in an online collaboration tool available to the agency partners and the project team.
- Rank Work Items - Agency partners rank their top X number of features. This is a financially unconstrained ranking since it is focused on what should be done in the next phase of development as opposed to how to do it. The top X features are selected for additional scoping refinement and each is assigned a point-of-contact.
- Refine Work Items - The point-of-contact refines the feature scope and works with the team to develop an approximate cost estimate. The refined feature descriptions with approximate cost estimates are discussed with the agency partners and refined again if necessary until acceptable.
- Allocate Funds - Each agency allocates the total phase of work budget to the features of interest, the funding allocation is discussed, and the agency partners select the final financially constrained list of features for the work plan.
- Finalize Work Plan - The team drafts a scope, schedule, and budget based on the final list of phase features and discusses with the agency partners. The scope of work is iterated with the project team until satisfactory.
- Execute - Upon execution of the phase scope of work, the feature point-of-contact continues in this role. The feature point-of-contact is responsible for coordinating the team and agency partner work on the feature. The entire project team may participate in the development of each feature as desired.

---

8 https://docs.google.com/spreadsheets/d/1wB_8HGpmtXL9vvysYdEItYyVMvw73r9FuRH3P-_bfEo/edit#gid=1486752204

The future ideas management and prioritization process, implemented with an online collaboration tool such as Google Docs or similar, should work well for managing future work ideas and prioritizing future work for each subsequent phase of the ActivitySim project. With that being said, the process and tools should be revisited before starting each subsequent phase of work scoping to ensure they continue to meet the needs.

Managing and Prioritizing Future Work Summary Table

Approach	**Brief Description**
Maintain a development roadmap	● Establish a development roadmap to help manage and prioritize future work
Phase work sprints	● Segment future work into discrete phases, each with a release
Maintain list of future work ideas	● Maintain a list of future work ideas in an online collaborative tools for everyone to contribute ● Coordinate with the third-party contributions list
Continue to implement a collaborative scoping effort	● Develop the phase scope, schedule, and budget with online collaborative tools in multiple rounds of brainstorming, describing, costing, and prioritizing
Establish a feature / task point-of-contact	● Appoint a point-of-contact, or sponsor, for each feature / task in the work phase to shepard the feature / task to completion

Summary

This strategic plan describes the foundation for the advancement of the ActivitySim platform as it transitions from its initial development to actual use for agency transportation planning studies. It covers the principles that guide ActivitySim development, as well as the methods for managing contributions from others, maintaining support for multiple model implementations, managing issues and requests, and managing ideas about future work items and prioritizing future work. With this

foundation, the platform and its leadership should be better prepared for ActivitySim's exciting future.