An Operational Open-Source Model System and Software Framework Supporting Agile Model Development of Strategic Planning Models

Brian Gregor, P.E.
Session 1C – Sketching a Vision with VisionEval
Innovations in Travel Modeling 2018
Atlanta, Georgia

Definitions

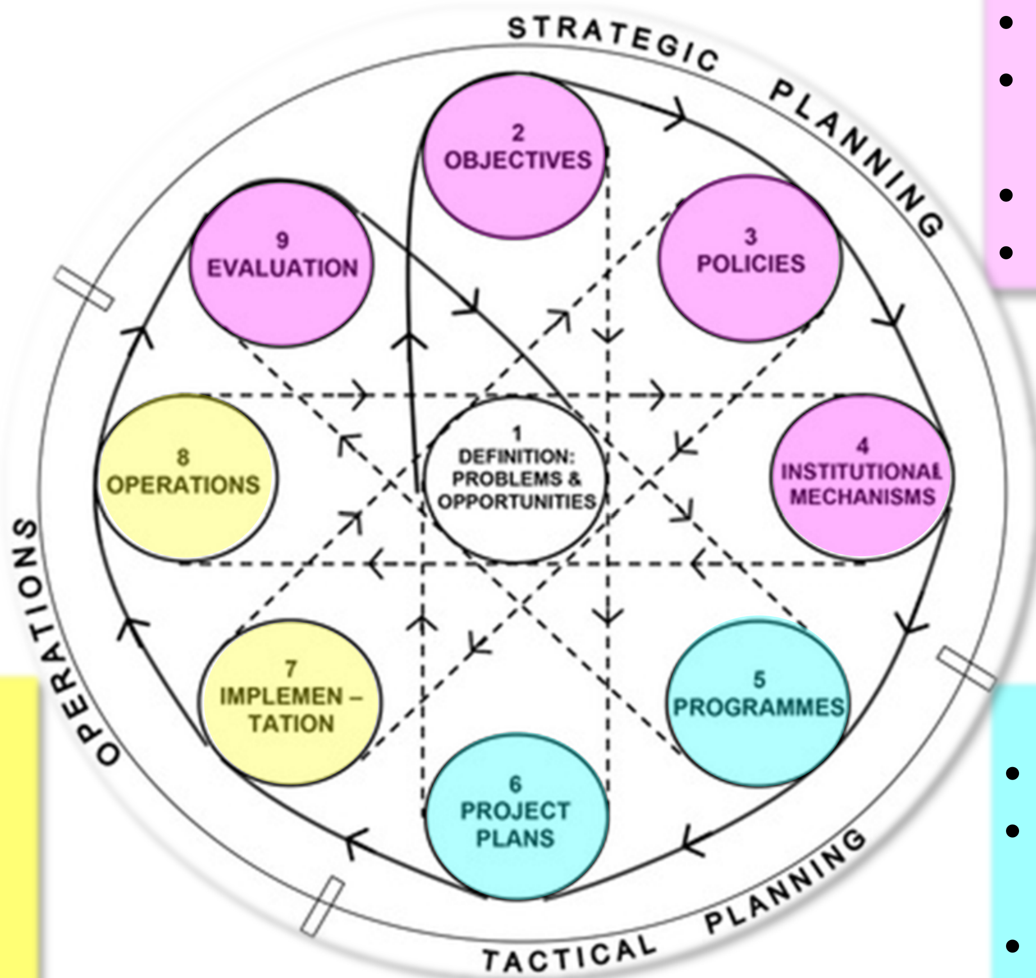
Model System: Specifications for modules that can be run in sequence to compose a model and a software framework for implementing those specifications. Models built in the model system are related by:

- The domains that the model system addresses
- The 'agents’ that are modeled
- How physical space is represented
- How time is represented
- Other modeling goals

Module: A module implements a sub-model, a component in a larger model (e.g. a trip generation sub-model in a 4-step travel demand model). A module includes a definition of the sub-model and the programming code to implement that definition.

VisionEval Modeling Domain: Strategic Visioning and Planning

Modified from planning diagram by:
Edward Leman (www.chreod.ca)



Strategic Planning Models
- Broad scope
- Limited detail (e.g. system level)
- Many scenarios
- **e.g. VisionEval**

Operations Models
- Limited scope
- Very detailed (e.g. intersection level)
- Few scenarios
- e.g. traffic simulation, transit operations

Tactical Models
- Moderate scope
- Moderate detail (e.g. link level)
- Few scenarios
- e.g. urban travel demand model

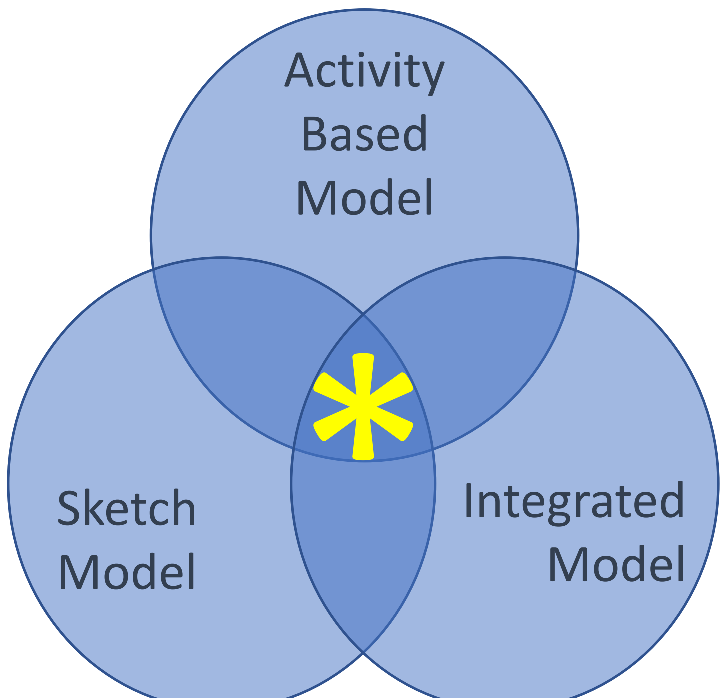
VisionEval Model Characteristics

VisionEval models address a broad range of considerations

- Household attributes
- Neighborhood characteristics
- Transportation system
- Prices and budgets
- Transportation operations
- Vehicles
- Fuels
- . . .

As with activity-based models, individual households are modeled but household behavior is modeled in aggregate

- Average DVMT
- Annual fuel
- Annual travel budget
- Annual walk trips
- . . .



Short run times enable 100s – 1000s of scenarios to be run

Budget models connect economics and travel

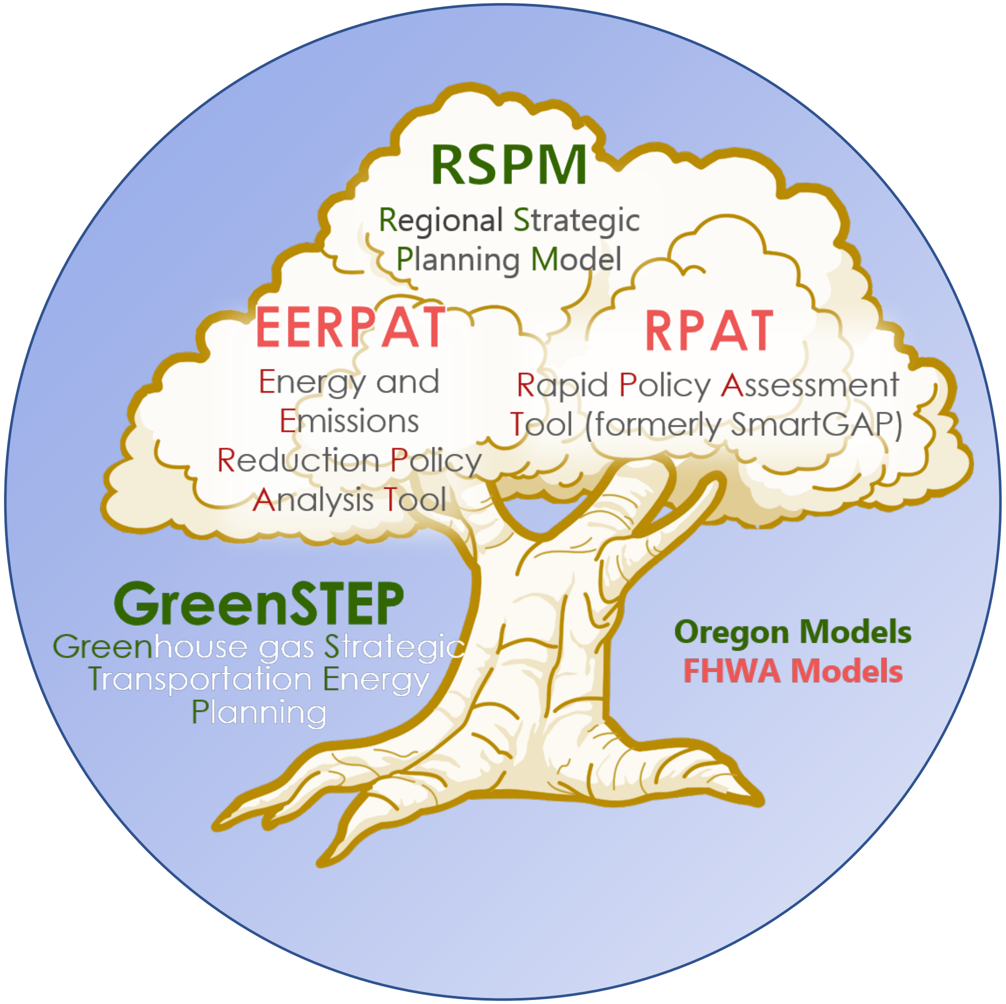
Choice models connect housing, location, and travel

Key VisionEval Model System Goals

Modularity: Structure model code so that it can be modified or replaced without requiring any modifications of other model code. Make modules interchangeable between models built in VisionEval model system.

Transparency: All aspects of modules and their implementation must be viewable, documented, and replicable.

Openness: The model system and software framework that supports it must have open, well documented standards and API with open-source licensing.



Benefits of Modular, Transparent, & Open System

Trustworthiness: People can examine how the models are estimated, how they are implemented in code and can run the code to test it. Data is documented. Data flows are traceable.

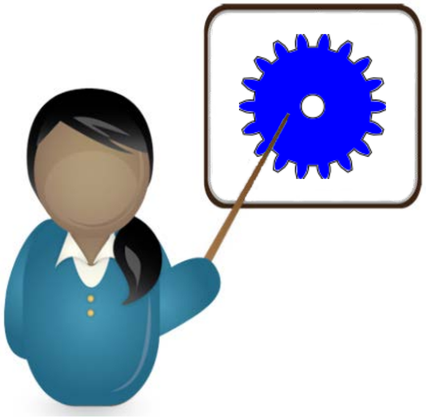
Reliability: Consistent standards and API make it easier to implement automated testing. Modules can be tested for consistency with standards. Module outputs can be tested for consistency with described behavior. Model inputs can be checked prior to running model.

Economical: Modules can be shared between models, reducing code duplication. Code maintenance is simplified.

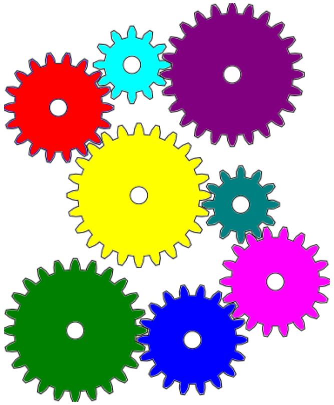
Agile Development: Modularity enables model development to be agile. A developer can start by implementing a minimal model and then incrementally improve the model to increase complexity, accuracy, and behavioral fidelity.

Connecting Research and Practice

The researcher focuses on an aspect of travel behavior or performance



To bring research into practice, it needs to be incorporated into a larger travel model

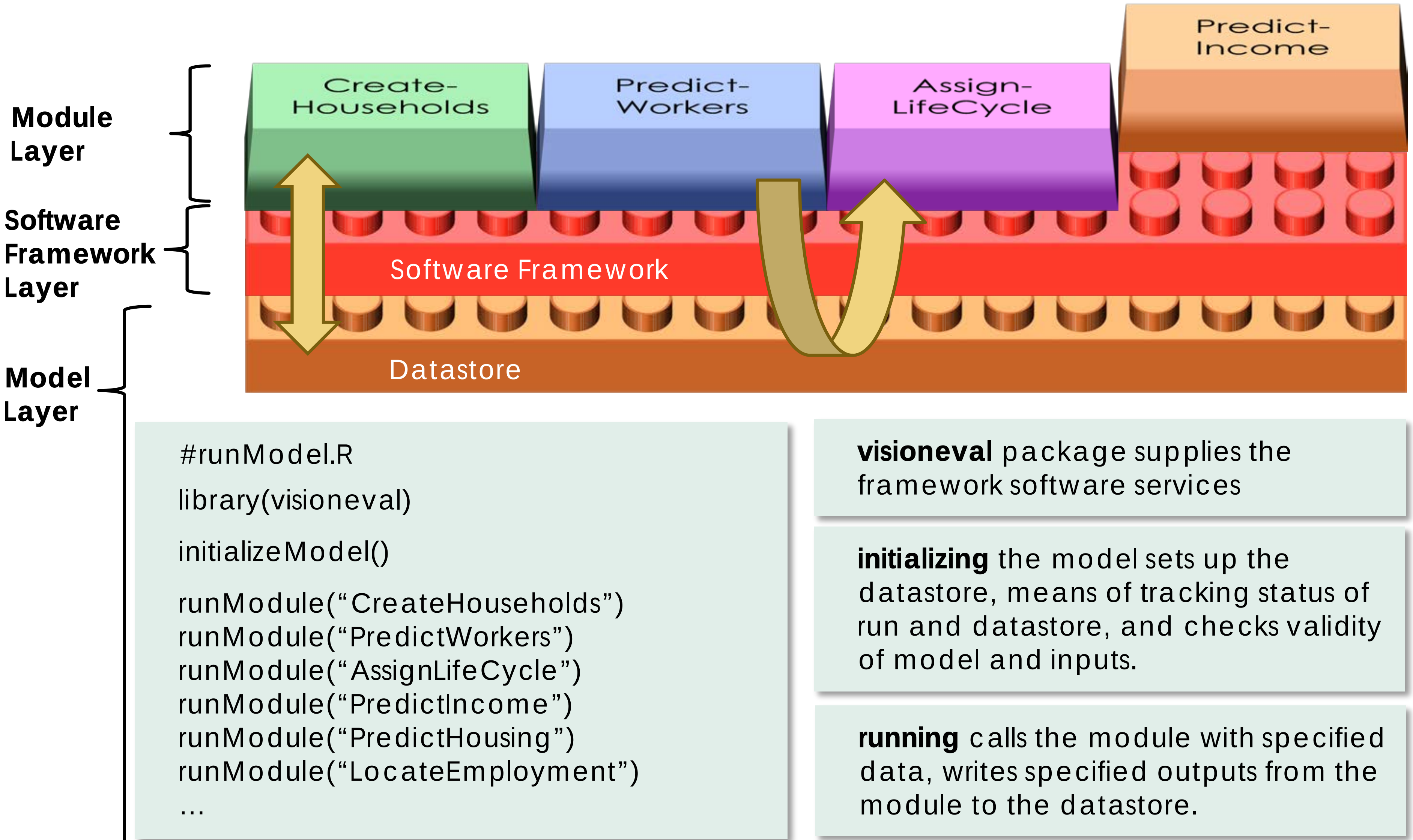


How can this be done?

Option	Issues
Add preprocessor or post-processor to existing model	Only works for research that can be accommodated at the front end or tail end of the model.
Build a model to plug the new component into	A lot of extra work for researcher. A lot of redundant work for many researchers.
Alter an existing model	Only works if the model includes complementary components and if the model and code are open, well documented and modular.

An open modular model system lowers the barriers to incorporating research into practice

VisionEval Model System in a Nutshell




```
#runModel.R
library(visioneval)
initializeModel()
runModule("CreateHouseholds")
runModule("PredictWorkers")
runModule("AssignLifeCycle")
runModule("PredictIncome")
runModule("PredictHousing")
runModule("LocateEmployment")
...
```

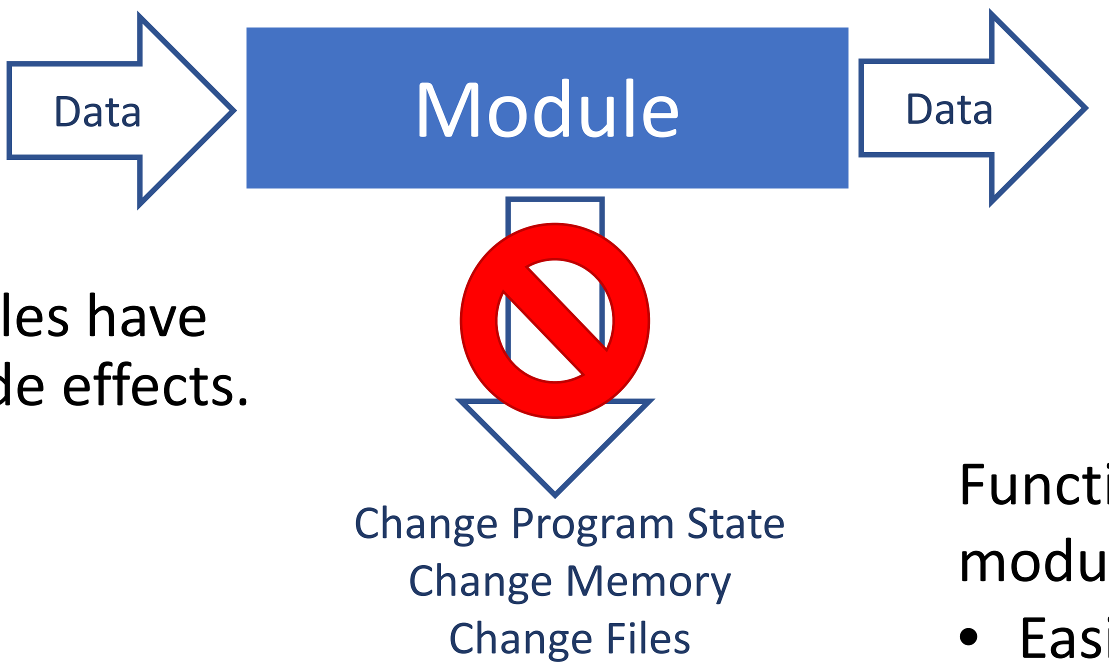
visioneval package supplies the framework software services

initializing the model sets up the datastore, means of tracking status of run and datastore, and checks validity of model and inputs.

running calls the module with specified data, writes specified outputs from the module to the datastore.

Functional Design for Modularity

Modules act like pure functions. They only consume and produce data.



Modules have NO side effects.

Functional approach makes modules and models:

- Easier to reason about
- More reliable
- Easier to test
- Easier to debug

Detailed Module Interface Describes All Resources Needed and Produced

Module interface components include:

- **RunBy**: *Level of geography module is to be run for*
- **NewInpTable**: *Identify tables that need to be created in the datastore to save module input data*
- **NewSetTable**: *Identify tables that need to be created in the datastore to save module output data*
- **Inp**: *Describe all data fields in user input files*
- **Get**: *Describe all datasets to be retrieved from the datastore to pass to the module*
- **Set**: *Describe all datasets that the module produces that are to be saved to the datastore*
- **Call**: *Identify other modules that will be called (more on this later)*

Detailed Data Specifications

Example of 'Set' specifications for two datasets produced by a module:

```
item(
    NAME =
      items("ComSvcDvmtPopulationFactor",
            "HvyTrkDvmtPopulationFactor"),
    TABLE = "Marea",
    GROUP = "Global",
    TYPE = "compound",
    UNITS = "MI/PRSN",
    NAVALUE = -1,
    PROHIBIT = c("NA", "< 0"),
    ISELEMENTOF = "",
    SIZE = 0,
    DESCRIPTION =
      items(
        "Ratio of base year commercial service
         vehicle DVMT to population",
        "Ratio of base year heavy truck DVMT
         to population"
      )
  )
```

Dataset names. Can list more than one dataset as long as other specifications are the same.

Table the datasets are to be placed in. There are tables for each level of geography as well as other categories of data such as Households.

Tables are organized in groups. The "Global" group contains data that is applicable to all model run years. If the GROUP value is "Year" the data is placed in the group for the model run year.

The data type and measurement units for the datasets. In addition to 4 primitive types, there are several complex types that define units and conversion factors between units. Compound types are made up of several complex types.

How NA values are stored.

Prohibited values if data is continuous. For categorical data, allowed values specified in 'ISELEMENTOF' attribute.

Maximum number of characters for string data.

Dataset descriptions.

Services Supported by Detailed Specifications

Check Model Consistency Prior to Runtime

Models are prechecked to determine whether each module that is called will be able to get the data it needs consistent with its specifications.

Check and Load User Inputs Prior to Runtime

All input data files are checked to determine whether the data are correct and complete. Modules may include scripts to do more complex input data checks and preprocessing.

Module Documentation and Cataloging

All input files and other data required by and produced by each module is documented in the module interface. This enables a module registry to be produced to assist model and module developers.

Automatic Unit Conversion

Standardized unit specifications for 13 complex data types (e.g. time, distance, mass, volume, energy) and the compound data type enable the framework to handle unit conversion. It also enables the framework to handle currency conversions to different years.

Interaction Between Modules is Controlled

- Controlling how modules can interact is important for module interchangeability and model reliability
- Primary interaction is through data exchange (to and from datastore) mediated by framework and governed by module interfaces
- Modules may call other modules to provide calculation services subject to the following limitations:
  - ‘Call’ specification must identify whether a module may be called
  - A module that may be called can not call any other module (avoids nested calls that reduce model comprehension and increase bugs)
  - A module that may be called can not have any user file inputs (avoids confusion for user)

Modules are Contained in R Packages

- **R has useful characteristics**
  - Excellent package management
  - Strong data science language
  - Facilitates open model development
  - Has interfaces to multiple languages
- **Related VE modules are usually combined in a package**
- **There is an R script for each VE module**
  - These are run when the package is installed
  - Model objects are created and saved
  - Functions are parsed
- **Packages include model estimation datasets unless they are large or confidential**
  - Regional data may be substituted for default to customize estimated parameters
  - Documentation for data is provided
- **Packages include module tests**

Example Source Package

```
│   DESCRIPTION
│   LICENSE
│   NAMESPACE
│   README.md
│
├───data
│
├───inst
│   │   NOTICE
│   │
│   └───extdata
│           pums_households.csv
│           pums_households.txt
│           pums_persons.csv
│           pums_persons.txt
│
├───man
│
├───R
│       AssignLifeCycle.R
│       CreateEstimationDatasets.R
│       CreateHouseholds.R
│       PredictIncome.R
│       PredictWorkers.R
│
└───tests
```

Example Module Script

```
#===================
#CreateHouseholds.R
#===================
##CreateHouseholds Module
#The CreateHouseholds module creates a *Household* table in the datastore
#and populates the table with datasets characterizing simulated households. Each
#entry represents a simulated household. Household datasets are created for the
#numbers of persons in each of 6 age categories (0-14, 15-19, 20-29, 30-54,
#55-64, and 65+) and the total number of persons in the household. Two types of
#households are created: *regular* households (i.e. not persons in group
#quarters) and *group quarters* households (i.e. persons in group quarters such
#as college dormatories). Households are created from Azone level demographic
#forecasts of the number of persons in each of the 6 age groups for *regular*
#households and for the group quarters population. In addition, users may
#optionally specify an average household size and/or the proportion of
#households that are single person households. The module creates households
#that matches the age forecast and the optional household size and single person
#inputs (close but not exact). The module tabulates the number of households
#created in each Azone.
#
###Model Parameter Estimation
#This model has just one parameter object, a matrix of the probability that a
#person in each age group is in one of several hundred *regular* household
#types. The matrix is created by selecting from the PUMS data the records for
#the most frequently observed household types. The default is to select the
#household types which account for 99% of all households. Each household type is
#denoted by the number of persons in each age group in the household. For
#example, a household that has 2 persons of age 0-14 and 2 persons of age 20-29
```

Example Module Script

```
#those households are just composed of single persons.
#
###How the Module Works
#For *regular* households, the module uses the matrix of probabilities that a
#person in each age group is present in the most frequently observed household
#types along with a forecast of number of persons in each age group to
#synthesize a likely set of *regular* households. The module starts by assigning
#the forecast population by age group to household types using the probability
#matrix that has been estimated. It then carries out the following interative
#process to create a set of households that is internally consistent and that
#matches (approximately) the optional inputs for household size and proportion
#of single-person households:
#
#1) For each household type, the number of households of the type is calculated
#from the number of persons of each age group assigned to the type. For example
#if 420 persons age 0-14 and 480 persons age 20-29 are assigned to household
#type *2-0-2-0-0-0*, that implies either 210 or 240 households of that type.
#Where the number of households of the type implied by the persons assigned is
#not consistent as in this example, the mean of the implied number of households
#is used. In the example, this would be 225 households. This is the *resolved*
#number of households. For all household types, the resolved number of
#households is compared to the maximum number of implied households (in this
#case 225 is compared to 240) if ratio of these values differs from 1 in
#absolute terms by less than 0.001 for all household types, the iterative
#process ends.
#
#2) If a household size target has been specified, the average household size for
#the resolved households is computed. The ratio of the target household size and
```

Example Module Script

```


#==============================================
#SECTION 1: ESTIMATE AND SAVE MODEL PARAMETERS
#==============================================
#This model has just one parameter object, a matrix of the probability that a
#person in each age group is in one of several hundred household types.
#Each household type is denoted by the number of persons in each age group in
#the household. The rows of the matrix correspond to the household types.
#The columns of the matrix correspond to the 6 age groups. Each column of the
#matrix sums to 1. The process selects the most frequently observed households.
#The default is to select the most frequent households which account for 99% of
#all households.

#Define a function to estimate household size proportion parameters
#--------------------------------------------------------------------
calcHhAgeTypes <- function(HhData_df, Threshold = 0.99) {
  Hh_df <- HhData_df[HhData_df$HhType == "Reg",]
  Ag <-
    c("Age0to14",
      "Age15to19",
      "Age20to29",
      "Age30to54",
      "Age55to64",
      "Age65Plus")
  #Create vector of household type names
  HhType_ <-
    apply(Hh_df[, Ag], 1, function(x)
```

Example Module Script

```

#Create and save household size proportions parameters
#-------------------------------------------------------
load("data/Hh_df.rda")
HtProb_HtAp <- calcHhAgeTypes(Hh_df)
#' Household size proportions
#'
#' A dataset containing the proportions of households by household size.
#'
#' @format A matrix having 950 rows (for Oregon data) and 6 colums:
#' @source CreateHouseholds.R script.
"HtProb_HtAp"
devtools::use_data(HtProb_HtAp, overwrite = TRUE)
rm(calcHhAgeTypes, Hh_df)


#==================================================
#SECTION 2: DEFINE THE MODULE DATA SPECIFICATIONS
#==================================================

#Define the data specifications
#------------------------------
CreateHouseholdsSpecifications <- list(
  #Level of geography module is applied at
  RunBy = "Region",
  #Specify new tables to be created by Inp if any
  #Specify new tables to be created by Set if any
  NewSetTable = items(
```

Example Module Script

```
#==================================================
#SECTION 2: DEFINE THE MODULE DATA SPECIFICATIONS
#==================================================

#Define the data specifications
#------------------------------
CreateHouseholdsSpecifications <- list(
  #Level of geography module is applied at
  RunBy = "Region",
  #Specify new tables to be created by Inp if any
  #Specify new tables to be created by Set if any
  NewSetTable = items(
    item(
      TABLE = "Household",
      GROUP = "Year"
    )
  ),
  #Specify input data
  Inp = items(
    item(
      NAME =
        items("Age0to14",
              "Age15to19",
              "Age20to29",
              "Age30to54",
              "Age55to64",
              "Age65Plus"),
      FILE = "azone_hh_pop_by_age.csv",
```

Example Module Script

```
#Specify input data
Inp = items(
  item(
    NAME =
      items("Age0to14",
            "Age15to19",
            "Age20to29",
            "Age30to54",
            "Age55to64",
            "Age65Plus"),
    FILE = "azone_hh_pop_by_age.csv",
    TABLE = "Azone",
    GROUP = "Year",
    TYPE = "people",
    UNITS = "PRSN",
    NAVALUE = -1,
    SIZE = 0,
    PROHIBIT = c("NA", "< 0"),
    ISELEMENTOF = "",
    UNLIKELY = "",
    TOTAL = "",
    DESCRIPTION =
      items(
        "Household (non-group quarters) population in 0 to 14 year old age group",
        "Household (non-group quarters) population in 15 to 19 year old age group",
        "Household (non-group quarters) population in 20 to 29 year old age group",
```

Example Module Script

```
#Specify data to be loaded from data store
Get = items(
  item(
    NAME = "Azone",
    TABLE = "Azone",
    GROUP = "Year",
    TYPE = "character",
    UNITS = "ID",
    PROHIBIT = "",
    ISELEMENTOF = ""
  ),
  item(
    NAME = "Marea",
    TABLE = "Azone",
    GROUP = "Year",
    TYPE = "character",
    UNITS = "ID",
    PROHIBIT = "",
    ISELEMENTOF = ""
  ),
  item(
    NAME =
      items("Age0to14",
            "Age15to19",
            "Age20to29",
            "Age30to54",
            "Age55to64",
            "Age65Plus"),
```

Example Module Script

```
  #Specify data to saved in the data store
  Set = items(
    item(
      NAME = "NumHh",
      TABLE = "Azone",
      GROUP = "Year",
      TYPE = "households",
      UNITS = "HH",
      NAVALUE = -1,
      PROHIBIT = c("NA", "< 0"),
      ISELEMENTOF = "",
      SIZE = 0,
      DESCRIPTION = "Number of households (non-group quarters)"
    ),
    item(
      NAME =
        items("HhId",
              "Azone",
              "Marea"),
      TABLE = "Household",
      GROUP = "Year",
      TYPE = "character",
      UNITS = "ID",
      NAVALUE = "NA",
      PROHIBIT = "",
      ISELEMENTOF = "",
      DESCRIPTION =
        items("Unique household ID",
```

Example Module Script

```
#=======================================================
#SECTION 3: DEFINE FUNCTIONS THAT IMPLEMENT THE SUBMODEL
#=======================================================
#This function creates households for the entire model region. A household table
#is created and this is populated with the household size and persons by age
#characteristics of all the households.

#Function that creates set of households for an Azone
#----------------------------------------------------
#' Create simulated households for an Azone
#'
#' \code{createHhByAge} creates a set of simulated households for an Azone that
#' reasonably represents a population census or forecast of persons in each of 6
#' age categories.
#'
#' This function creates a set of simulated households for an Azone that
#' reasonably represents the population census or forecast of persons in each of
#' 6 age categories: 0 to 14, 15 to 19, 20 to 29, 30 to 54, 55 to 64, and 65
#' plus.
#'
#' @param Prsn_Ap A named vector containing the number of persons in each age
#'   category.
#' @param MaxIter An integer specifying the maximum number of iterations the
#' algorithm should use to balance and reconcile the population allocation to
#' household types.
#' @param TargetHhSize A double specifying a household size target value or NA
```

Example Module Script

```
#Main module function that creates simulated households
#------------------------------------------------------
#' Main module function to create simulated households
#'
#' \code{CreateHouseholds} creates a set of simulated households that each have
#' a unique household ID, an Azone to which it is assigned, household
#' size (number of people in the household), and numbers of persons in each of
#' 6 age categories.
#'
#' This function creates a set of simulated households for the model region
#' where each household is assigned a household size, an Azone, a unique ID, and
#' numbers of persons in each of 6 age categories. The function calls the
#' createHhByAge and createGrpHhByAge functions for each Azone to create
#' simulated households containing persons by age category from a vector of
#' persons by age category for the Azone. The list of vectors produced by the
#' Create Households function are to be stored in the "Household" table. Since
#' this table does not exist, the function calculates a LENGTH value for the
#' table and returns that as well. The framework uses this information to
#' initialize the Households table. The function also computes the maximum
#' numbers of characters in the HhId and Azone datasets and assigns these to a
#' SIZE vector. This is necessary so that the framework can initialize these
#' datasets in the datastore. All the results are returned in a list.
#'
#' @param L A list containing the components listed in the Get specifications
#' for the module.
#' @return A list containing the components specified in the Set
#' specifications for the module along with:
```

Example Module Script

```r
#' @export
CreateHouseholds <- function(L) {
  #Define dimension name vectors
  Ap <-
    c("Age0to14", "Age15to19", "Age20to29", "Age30to54", "Age55to64", "Age65Plus")
  Ag <- paste0("Grp", Ap)
  Az <- L$Year$Azone$Azone
  #fix seed as synthesis involves sampling
  set.seed(L$G$Seed)
  #Initialize output list
  Out_ls <- initDataList()
  Out_ls$Year$Azone$NumHh <- numeric(0)
  Out_ls$Year$Household <-
    list(
      Azone = character(0),
      Marea = character(0),
      HhId = character(0),
      HhSize = integer(0),
      HhType = character(0),
      Age0to14 = integer(0),
      Age15to19 = integer(0),
      Age20to29 = integer(0),
      Age30to54 = integer(0),
      Age55to64 = integer(0),
      Age65Plus = integer(0)
    )
  #Make matrix of regular household persons by Azone and age group
  Prsn_AzAp <-
```

Other Features

- Logical design of datastore and the software interface enables alternative implementations
  - Column-oriented table structure to support vectorized calculations
  - Current implementations include HDF5 and R binary files
- Module testing functionality
  - Module developers can test their module during the development process
  - Tests check whether
    - The module interface is correct
    - The test input files are consistent with interface specifications
    - Test datastore data are consistent with interface specifications
    - Module outputs are consistent with interface specifications
  - Module test data and scripts are included in the package
- A model run can reference one or more datastores
  - Facilitate scenario management
  - Facilitate consistency in model applications

Status

- VisionEval model system design and software framework package (visioneval) are complete
- RPAT and RSPM models have been converted to VisionEval and work is underway to convert the GreenSTEP model
- Work is underway to develop:
    - A full-featured graphical user interface
    - Methods and tools to set up and run large numbers of scenarios
    - Data visualizer to investigate the results of large numbers of scenarios
- A software management and testing process has been set up (Ben Stabler presentation)
- A new multi-modal travel module has been developed (Liming Wang presentation)
- A pooled fund project to support further development has been set up (Dan Flynn presentation)

Thank You

Brian Gregor
Oregon Systems Analytics
gregor@or-analytics.com
https://github.com/gregorbj

Acknowledgements:

- Oregon Department of Transportation
- Federal Highway Administration Office of Planning, Environment, and Realty
- American Association of State Highway and Transportation Officials
- Volpe National Transportation Systems Center

Pooled Fund (FHWA-Volpe)

Participants

DOTs	MPOs
▪ OR	▪ Las Vegas
▪ MD	▪ Atlanta
▪ WA	▪ Houston
▪ Ohio	
▪ NC	
▪ **CA**	

visioneval.org

Jeremy Raw, FHWA jeremy.raw@dot.gov