METHODS REPORT

Sharing Research Models: Using Software Engineering Practices for Facilitation

Stephanie P. Bryant, Eric Solano, Susanna Cantor, Philip C. Cooley, and Diane K. Wagener

March 2011

RTI Press

RTI INTERNATIONAL

About the Authors

Stephanie P. Bryant, MS, is a member of the Bioinformatics Group at RTI International, specializing in software development, distributed systems, and high-performance computing.

Eric Solano, PhD, is a member of RTI's Bioinformatics Group, specializing in high-performance computing tools (distributed computing, grids) in simulations, modeling, and programming.

Susanna Cantor, BS, a member of RTI's Research Computing Division, specializes in system, software, and project documentation, and process and configuration management.

Philip C. Cooley, MS, is an RTI Fellow in bioinformatics and high-performance computing.

Diane K. Wagener, PhD, is a member of the Genomics, Statistical Genetics, and Environmental Research group at RTI International. Her research includes genomics and proteomics of vaccine response and informatics support for modeling the spread of infectious disease.

RTI Press publication MR-0022-1103

Suggested Citation

Bryant, S. P., Solano, E., Cantor, S., Cooley, P. C., and Wagener, D. K. (2011). Sharing Research Models: Using Software Engineering Practices for Facilitation. RTI Press publication No. MR-0022-1103. Research Triangle Park, NC: RTI Press. Retrieved [date] from http://www.rti.org/rtipress.

This publication is part of the RTI Press Methods Report series.

RTI International
3040 Cornwallis Road
PO Box 12194
Research Triangle Park, NC
27709-2194 USA

Tel: +1.919.541.6000
Fax: +1.919.541.5985
E-mail: rtipress@rti.org
Web site: www.rti.org

doi:10.3768/rtipress.2011.mr.0022.1103

www.rti.org/rtipress

Sharing Research Models: Using Software Engineering Practices for Facilitation

Stephanie P. Bryant, Eric Solano, Susanna Cantor, Philip C. Cooley, and Diane K. Wagener

Abstract

Increasingly, researchers are turning to computational models to understand the interplay of important variables on systems' behaviors. Although researchers may develop models that meet the needs of their investigation, application limitations—such as nonintuitive user interface features and data input specifications—may limit the sharing of these tools with other research groups. By removing these barriers, other research groups that perform related work can leverage these work products to expedite their own investigations. The use of software engineering practices can enable managed application production and shared research artifacts among multiple research groups by promoting consistent models, reducing redundant effort, encouraging rigorous peer review, and facilitating research collaborations that are supported by a common toolset. This report discusses three established software engineering practices—the iterative software development process, object-oriented methodology, and Unified Modeling Language—and the applicability of these practices to computational model development. Our efforts to modify the MIDAS TranStat application to make it more user-friendly are presented as an example of how computational models that are based on research and developed using software engineering practices can benefit a broader audience of researchers.

Contents

Introduction 2

Methods 2

General Software Development Tasks 2

Infectious Disease Models 3

TranStat Application for Infectious Disease Models 3

Software Engineering Practices 3

The TranStat Modification 6

Limitations of the Initial TranStat Application 6

Modification of the TranStat Application 7

Using Software Development Phases to Extend Program Features 9

Discussion 13

References 14

Acknowledgments Inside back cover

RTI INTERNATIONAL

Introduction

Biological and medical research has a long history of modeling complex systems to understand the impacts of important variables on systems' behaviors. Traditionally, this research has involved mathematical modeling; however, computer modeling (i.e., simulation) is increasingly being used to understand systems that either are too complex to have closed-form analytic solutions or are dynamic systems in which the behavior of a system changes over time. Increasing model complexity has fostered the sharing and joint development of models by multiple, independent research groups to leverage these work products and expedite investigations. Examples of this trend are the Models of Infectious Disease Agent Study (MIDAS), funded by the National Institute of General Medical Studies,[1] and the Program in Systems Immunology and Infectious Disease Modeling, funded by the National Institute for Allergy and Infectious Diseases.

The development and evolution of computer models involves dynamic model expansion and modification to address new scientific questions; therefore, effectively sharing models among investigators can be a challenge. Because model-specific limitations, such as nonintuitive user interface features and data input specifications, may limit ease of use by researchers who are unfamiliar with an application, models created for a specific research study are rarely used outside the originating research group.

In this report, we describe standard software engineering practices that researchers can adopt during the model development process to increase their usability by external research groups. To support this viewpoint, we provide an example of our work modifying TranStat, an application developed by researchers from one of the MIDAS groups. These practices are widely applicable to other fields of research.

Methods

General Software Development Tasks

The software development process should follow a standard product development method. Several recognized software engineering practices can benefit the development and implementation of computational models by promoting standard development procedures that encompass the development process from design to deployment. The use of standard documentation tools in particular can facilitate wider model distribution among research groups.

In general, the software development process life cycle typically has five phases:

- **Requirements phase**— Gather information detailing the requisite features and use cases of the software product based on the problem statement.
- **Design phase**—Extract implementation constructs from the requirements documentation.
- **Implementation phase**—Implement constructs identified in the design phase in the target programming language.
- **Testing phase**—Execute the software product to ensure that it exhibits the proper behavior as specified in the requirements.
- **Deployment phase**—Transition the successfully tested product to its production environment.

By using software engineering practices that emphasize modular design and implementation to allow for software reuse, extensibility, and complexity management, researchers can develop systems that are easier to maintain internally. They can also develop more robust and usable computational modeling solutions that are useful beyond the originating research group, thereby supporting collaborative efforts. Employing well-established and accepted industry analysis and implementation practices can facilitate this type of model development: three important practices are use of an iterative software development process (ISDP), object-oriented methodology (OOM), and Unified Modeling Language (UML). We discuss these in later sections of this report.

Infectious Disease Models

MIDAS researchers develop "computational models of the interactions between infectious agents and their hosts, disease spread, prediction systems, and response strategies."[1] The models are useful resources for policymakers, health workers, and other researchers in understanding and developing responses to infectious diseases. In the event of an infectious disease outbreak, policymakers may engage the MIDAS group to develop specific models that characterize the situation and provide additional assessment information. Research groups funded by the program include a consortium of universities and not-for-profit institutions.

TranStat Application for Infectious Disease Models

Developed as a part of the MIDAS effort, the TranStat application is the result of research by scientists from the University of Washington and the Fred Hutchinson Cancer Research Center (hereafter UW/FH).[2] TranStat is a software application used to analyze data from acute disease outbreaks, examine transmission characteristics, and estimate epidemiological characteristics of a disease.[3] The computational model part of the application explores how the contact between people in a population affects transmission of a disease.

The input set of the model includes descriptions of the population members, contact patterns, infection risk specifications, and the characteristics of the proposed infection, for example, probability distributions for the incubation and infectious periods. The model produces estimates of the length and outcomes of the infectious disease outbreak. In addition, the output set includes values for widely accepted epidemiological metrics, such as secondary attack rates within and between households, probability of infection, and case fatality rate. This model is useful for rapidly investigating infection disease outbreak scenarios.

TranStat had originally been developed to support investigations of cases of avian influenza. It provided mechanisms for data entry, storage, and analysis of disease outbreak data. Investigators can use the data to analyze acute disease outbreaks, examine transmission characteristics, and estimate the epidemiological characteristics of a disease.[3] Originally, however, it had not been designed for extensibility (i.e., ability to be expanded to support new or improved features). Some features were hard-coded and could be changed only by editing and recompiling the source code. The validation scheme did not easily support custom configurations. The initial version also did not easily support dynamic graphical information display. Applying TranStat to other research pursuits required additional features that were not originally supported. Addressing these constraints could promote use by other researchers, and this was a major goal of the RTI team.

The revised TranStat application has a modified user interface, adding an enhanced graphical user interface (GUI), input validation, extended help features, and graphical results output. The application is deployed as a precompiled software package and executed as a stand-alone PC-based computer application. It is offered publicly through the MIDAS web portal.[4]

Specifically, TranStat now provides a GUI implemented in Java (an object-oriented programming language) for analysts to enter and manipulate observed subject-to-subject contact information and disease transmission characteristics. The main interface uses tabbed pages to segment the data input into manageable sections and guide the user through the process of using the model. The input information is validated and used by the computational model to calculate the outbreak outcomes and epidemiological metrics.

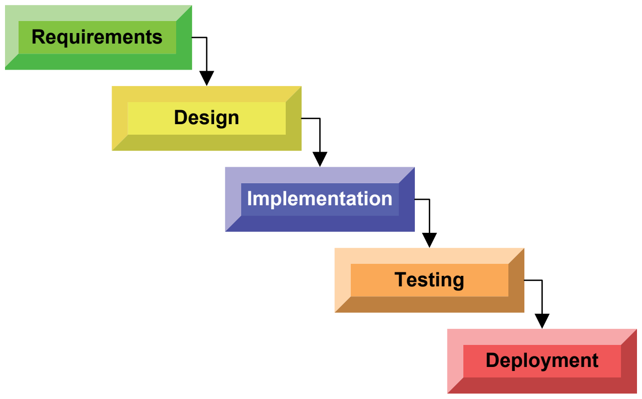
Data analysis is performed by a computational model of infectious disease propagation through a population, which is implemented in C programming language. TranStat displays analysis results in tabular and graphical formats. The application contains descriptive assistance features to explain input and results information.

Software Engineering Practices

Waterfall Software Development Process

A well-known software development process is the waterfall process (Figure 1). In this, developers perform the five software development phases sequentially, with the output of one phase serving as input to the next phase. All input information for a specific phase is available to researchers while performing activities within that phase; however, once completed, developers do not revisit previous phases.

Figure 1. Waterfall software development process

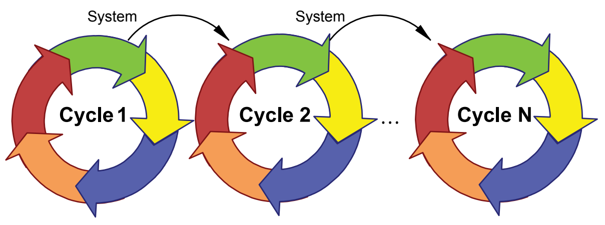


This is, however, an idealized model. In reality, not all the information needed for a specific phase may immediately be known or available; moreover, an application's parameters may change as a result of a phase's outputs. Thus, the waterfall process does not necessarily support the realities of actual software development and may not be the most efficient process for certain projects.

Iterative Software Development Process

In contrast to the waterfall process, the iterative software development process (ISDP, see Figure 2) allows for flexibility during software development. In this process, developers conduct multiple cycles of the waterfall process before delivering the final software product.[5] At the conclusion of each cycle, developers produce an interim or incremental software product that incorporates a subset of the requirements to be tested and evaluated. Each interim product serves as additional input for the next development cycle. Thus, developers can revisit the development phases; this capability allows for sufficient review, management, and incorporation of any needed software changes and yields a product that evolves over time using multiple evaluations.

Figure 2. Iterative software development process



Object-Oriented Methodology

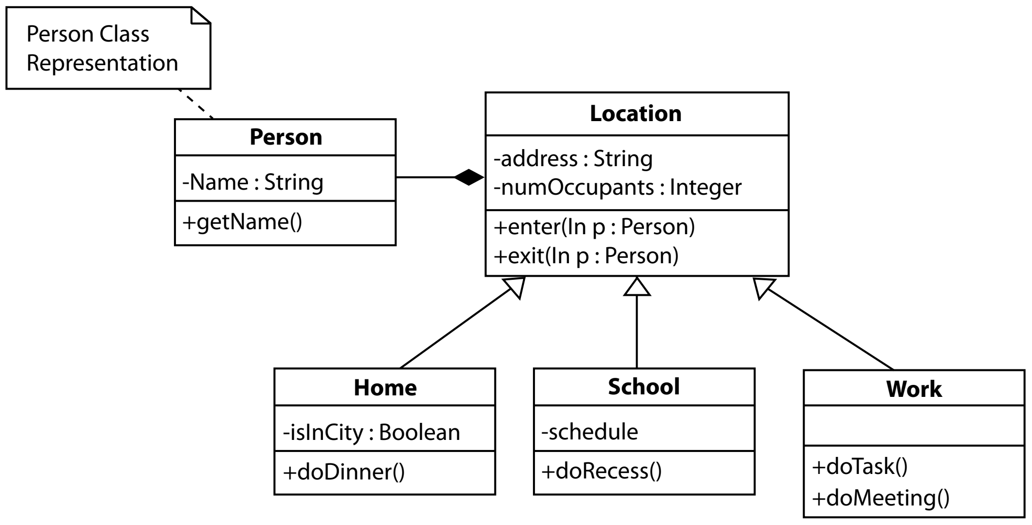
Object-oriented methodology (OOM) is a software development ideology that enables the creation of software programs that utilize object-oriented module units that are adaptable for use in other programs. The basic unit in OOM is a *class*,[6] which is a flexible construct that may represent concrete or abstract concepts in the problem space.

Classes are template specifications that—during execution of an object-oriented application—serve as the basis for creating one or more distinct copies or instances, called *objects*. A class contains methods (i.e., groupings of software instructions) and data members, called *class variables* (i.e., named memory locations shared by the methods) in an access-controlled unit. OOM class diagrams show the class structures and the hierarchical, aggregation, association, and dependency relationships among classes. Figure 3 represents an OOM class diagram in which classes are templates that describe specific elements in the problem domain.

Figure 3 represents a scenario in which people move among different locations and interact with one another. The relevant components (i.e., *Person, Location, Home, School, Work*) are separate classes. Within a single application execution, multiple instances of classes are created. For example, the *Person* class is the template specification for creating multiple *Person* objects. Each *Person* object contains its own copy of class variables whose data values can be different without affecting other *Person* objects.

Sets of classes and their relationships can express increasingly complex relationships and functionality while serving as an organized framework through which developers can manage the complexity of the software system. OOM supports several types of

Figure 3. Object-oriented methodology concepts



relationships among classes, including **inheritance** and **composite** relationships. The inheritance relationship is similar to a parent-child relationship, in which derived classes inherit methods and variables from their base class. Inheritance enables a derived class to reuse methods from its base class without duplicating the method instructions in the derived class.

In Figure 3, the derived classes *Home*, *School*, and *Work* are derived from the base class, *Location*, whose methods and data are available to be reused by the derived classes. The derived classes can also extend the base-class functionality by defining additional class variables and methods.

The composite relationship defines a containment relationship where the containing class specifies references or handles to other classes. In Figure 3, for instance, a *Location* may contain references to zero or more *Person* objects represented in a population. The example design shown in Figure 3 can be robust enough to support populations of varying sizes, from tens to thousands of people. For many software implementations, the size of the population being represented is limited only by the physical memory of the available computing resources.

Communication and interaction in a class are represented by specifying an appropriate invocation signature. Specifying a method invocation means including the name of a method and requisite parameters within another method's set of instructions. In Figure 3, interactions and communication patterns among the classes are represented by invocations of methods, such as *doDinner()*, *doRecess()*, and *doMeeting()*.These invocations are shown with parentheses to indicate that they can accept the input parameters used by the methods' instructions.

During execution of an object-oriented application, distinct instances of a class are called *objects*. Within a single execution, multiple instances of classes may be created, for example with the *Person* class. Each object has an independent set of data variables. During execution, *School* objects may have different values for the data members.

OOM supports the software development process by specifying the approach developers should use to analyze, design, implement, and validate an object-oriented software system:

- **Requirements phase**—Analyze the problem statement to identify the concepts in the problem domain and the functionality to be represented in the software system.
- **Design phase**—Represent the set of identified concepts using object-oriented notations, and refine the set of classes based on use cases identified in the requirements phase.

- **Implementation phase**—Express the object-oriented representations in the classes and extract relationships between entities in the problem space; implement class structures in a target object-oriented programming language.
- **Validation and testing phases**—Ensure that the implementation delivers the requisite functionality.

Using OOM, researchers can develop modular applications that are easier to reuse or maintain than those derived from monolithic programs (i.e., a program that contains a single function for the entire program). Because the related functionality is grouped into the class modules, a class can be used many times within an application without duplicating the software instructions. This extends the reuse of the application and reduces maintenance because it is not necessary for developers to make the same change in multiple places in the application. In a design based on OOM, a class can encapsulate a set of complex software instructions or an algorithm and reduce the algorithm to one software instruction.

Unified Modeling Language

Software object-oriented designs vary in both their complexity and their platform and language options for implementation. Unified Modeling Language (UML) is a common language for expressing and sharing software object-oriented designs; it is independent of a software's programming language and platform.

The OOM class diagram outlined in Figure 3 is presented in UML notation. For instance, the open triangles represent the inheritance relationship among the classes, the closed diamond represents a composite relationship, and the boxes represent a class. UML diagrams capture both the static (class) structure of the system (captured by various relationships and dependencies among classes) and the dynamic behavior (objects and their behavior while the application is running); they also capture deployment scenarios.

Because UML is a standard notation for visually specifying and documenting a software system,[7] the language encompasses various model diagrams suitable for system specification by a variety of participating practitioners, from the software architect to the software implementer. In addition, many existing documentation applications, such as Microsoft Visio and other software documentation tools, support UML notation.

The TranStat Modification

As noted above, as part of the MIDAS effort, UW/FH investigators (the originating research group) developed the first TranStat application. TranStat was developed to support investigations of cases of avian influenza and was not designed (initially) to be extended beyond this purpose.[2]

The core of TranStat is a computational model that explores how contact among people in a population affects the transmission of a disease. The input set of the model includes descriptions of the population members, contact patterns, infection risk specifications, and characteristics of the proposed infection, such as probability distributions for the incubation and infectious periods. The model produces an output set that estimates the length and outcomes of the infectious disease outbreaks and includes values for widely accepted epidemiological metrics, such as secondary attack rates within and between households, as well as rates for probability of infection and fatalities.

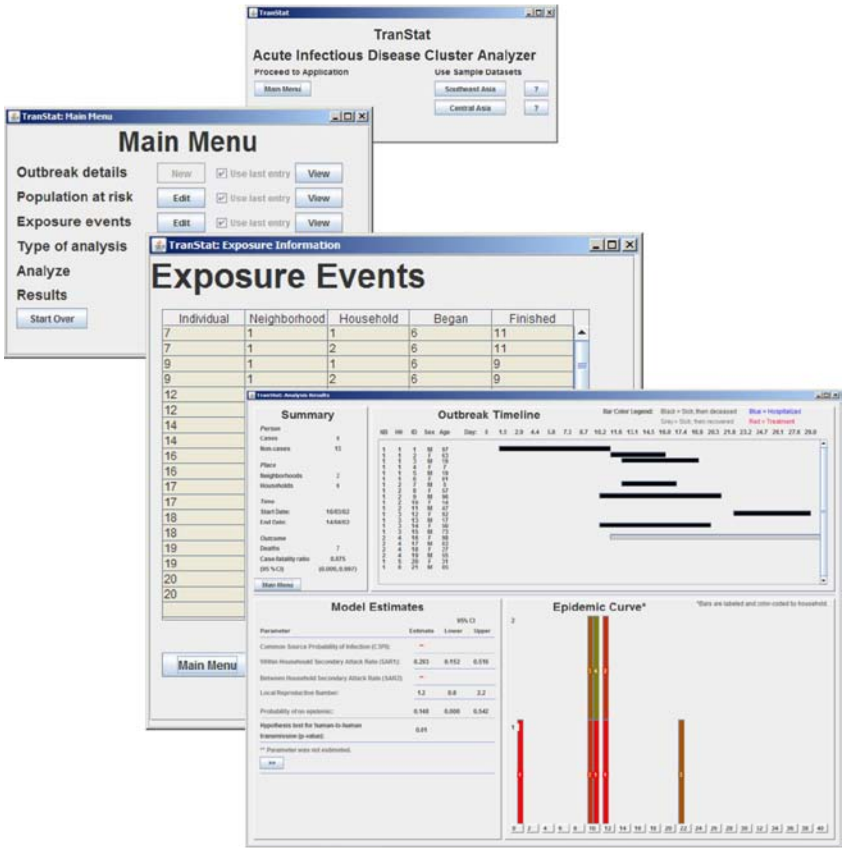
Limitations of the Initial TranStat Application

Several application issues limited the use of the initial TranStat application by external research groups for other research pursuits. The GUI of the initial application consisted of a set of independent screens that appeared based on user selection. These screens provided little guidance regarding interaction and navigation expectations to those unfamiliar with the application.

The original input data validation scheme did not easily support custom configurations. Also, the initial TranStat application limited the data input to a fixed amount. Using the application in other efforts, which may require more input information, called for removing the constraints on the amount of input data specified in the application. Also, the validation scheme had to be able to check the limits of each data item in the input set, even when the total number of data items was not known in advance.

Finally, the initial TranStat application did not easily support a dynamic graphical information display. Information displayed in the graphs of the initial application was generated based on the input parameters to the computational module, as shown in Figure 4. Because the graphical display used fixed positioning to display result information, when the display exceeded the statically allotted display area, the graph was cut off; no options for providing additional scroll bars were available.

Figure 4. Sample screens of initial TranStat application



Individual	Neighborhood	Household	Began	Finished
7	1	1	6	11
7	1	2	6	11
9	1	1	6	9
9	1	2	6	9
12				
12				
14				
14				
16				
16				
17				
17				
18				
18				
19				
19				
20				
20				

Modification of the TranStat Application

The development and extension of features to modify the TranStat application was performed jointly by the UW/FH group and the software engineering team at RTI International. Strong communication between the geographically distributed collaborating teams was essential in achieving the goal of enhancing the TranStat application, and the teams used multiple forms of communication, including e-mail, telephone conference calling, and in-person meetings, to facilitate this work.[8]

Each participating team had a slightly different perspective on the application modification. For example, the UW/FH group had concerns about maintaining the appropriateness of the revised application's output, whereas the RTI team was focused on improving the application's robustness and user interface aspects.

To establish a common direction for this collaborative development effort, the teams used the ISDP to identify and refine the main tasks for the modification. Determining the scope of the modification consisted of examining the initial TranStat application, identifying suitable areas for modification, and designing modifications that accommodated the computational model. In addition, overcoming modification challenges required a clear understanding of the dynamics of gathering and preparing the input data for use by the computational model and refining the requirements to address potential concerns of a broader community of users.

The teams concluded that modifying TranStat should involve the following three steps:

1. Modify the GUI of the initial application with features that could support a wider set of potential users.
2. Extend the data handling capabilities.
3. Improve the application's data visualization capabilities.

Modifying the Graphical User Interface

The changes we made to refine the initial application ranged in complexity of implementation. Extending the GUI was the most complex task. It required considerable communication and input from members of both teams. We needed to work closely with the originating research group to capture their expertise in using the initial application and to translate that knowledge into graphical representations.

For example, the GUI requirements for TranStat either were highly focused on the research group's recent investigation or were fairly abstract because of the absence of insight as to how other researchers might use the application. In addition, because the GUI was not designed to be used beyond the originating research group, its design had not supported addition of new features or extension of existing features.

The teams addressed these identified constraints at key stages in the ISDP and used them as a basis for developing a modular and flexible interface solution that was compatible with the existing computational model. That is, we implemented modified features according to an extensible object-oriented software design using accepted software implementation practices. The RTI team used the Jigloo GUI library[9]—a commercially available resource—that met both teams' expectations and the anticipated needs of other users. The GUI now supports more dynamic data entry and parameter customization. In addition, we used a tabbed page layout to segment the data entry and analysis features.

The RTI team applied OOM techniques to analyze the initial program structure and develop a modular design that contained OOM relationships among the indentified constructs. The user interface of the initial TranStat application was implemented as separate windows; however, for the revised design, the team extracted and captured the common functionality of each individual window into an OOM base class that could be reused. In addition, the team captured the specialized features of the segmented data windows in derived classes, which extended the functionality of the base class. Finally, the RTI team decided to use a tabbed container as the primary user interface construct. This container enabled the team to incorporate consistent display elements, such as the layout features, help buttons, and navigational buttons.

In short, the revised design combines common functionality and specialized functionality into a manageable and extensible GUI. The initial application has been extended with an revised user interface and supporting features that may increase the model's use by other external researchers.

Extending the Data Handling Capabilities

The development effort we used established software development practices supported by an iterative software development process. We enhanced the user experience by providing ordered data entry pages, segmenting data entry and analysis features into manageable groupings. The model results are shown as formatted text and graphics, accompanied by expanded explanatory labeling. We extended the data validation scheme to ensure data integrity prior to data analysis.

RTI staff obtained the requirements for extending TranStat's data from the UW/FH research group; we focused on data import, validation, and storage. Data import addresses the manner in which model configuration values residing in external text files are transferred from the text files to the application's memory space. During data transfer, data validation checks the data values to ensure that they are within acceptable limits; once this step is completed, the application stores all validated data in its active memory.

Extending the data input and validation features was a moderately complex task with a wide scope. The initial TranStat application had provided a limited description of the data format to those unfamiliar with the design of the computational model. In addition, the arrangement of the data in memory had not been conducive to easy access by all program segments requiring the information.

The amount of data required to describe a target population can be considerable. Because the size of the population and the supporting data are not known until runtime, the design of the initial TranStat application had not been flexible enough to support data of varying complexity.

The RTI team used OOM to analyze the data import requirements of the application and convey the lessons and experience gained by the UW/FH research team through the course of their investigation within a flexible data import algorithm that could support the input of data of varying complexity. Before beginning work, we consulted members of the UW/FH group to understand the purpose and limits of each data item in the external text file. Then the RTI team examined the initial implementation to determine common features of the requisite processing steps. We used OOM techniques to partition the data into appropriate groupings that enabled application components to access data reusing the same algorithm. The revised application's validation algorithm validated all data before a user was permitted to initiate the data analysis.

Improving Data Visualization Capabilities

Adapting the visual presentation of some computed results was a low-complexity task. It involved ensuring that the proper output information was available and correctly formatted for use by the classes in the data visualization library.

In accordance with the reuse strategy, the RTI team used the JFreeChart[10] data visualization library to produce graphs and charts of the calculated epidemic curve and outbreak timelines for the input data. JFreeChart is an open-source and object-oriented data visualization library written in the Java programming language. The team integrated the data visualization package into the revised TranStat application. Because the TranStat application and the JFreeChart visualization library were written in the same object-oriented programming language, this effort entailed making the appropriate data available to the data visualization library and invoking the proper library methods for the graphing package to display the data in chart or graph form with explanatory labeling.

Testing

In the revised application, the validation algorithm checks the data items, provides assistive messages (i.e., dialog boxes) when data items are outside of expected limits, and logs entries that identify the data item that failed validation. The assistive messages indicate the location of the data item within the input file that failed validation, the expected value, and the purpose of the data item. The validation algorithm also propagates changes to data values to all displays so that they display all the latest validated information.

Using Software Development Phases to Extend Program Features

The RTI team used ISDP phases to support the development and extension of features to produce a more refined version of TranStat. ISDP aided the team's work by establishing a common direction for the collaborative development effort between UW/FH researchers and the software engineering group at RTI.

With ISDP as a guiding framework and key collaborative tool, we were able to use the process phases to identify tasks jointly and to focus our efforts in a common direction. The design phase entailed examining the application's original state, identifying suitable areas for modification, and designing modifications that accommodated the existing program structures. The RTI team implemented the changes according to the modification designs. The RTI team performed initial testing using input data sets provided by the UW/FH group. Later, for objective implementation review, separate testing personnel from RTI tested the application. During the development phase, RTI staff tested interim versions of the application and the UW/FH group reviewed them.

After both groups determined that the set of requisite features was present and that the output results of several testing situations were consistent with the UW/FH research group's expected model results, the RTI team released the application to the public by making it available through the MIDAS web portal for download. The portal offers users the ability to submit comments and raise issues regarding the new release.

Customized Iterative Software Development Process

Applying orderly software engineering practices to a rapidly evolving software that is focused on the computational modeling of dynamic systems can be challenging because the computational model must incorporate new discoveries or observations as they are made. Thus, the dichotomy between control for the purpose of stability and change influenced by discovery is ever-present. To account for this, the RTI team worked closely with the researchers to understand or develop their requirements while also considering these requirements and supporting implementation in a broader context for other users.

At key points in each development cycle of the iterative process used for the TranStat modification, the teams established a nonmodifiable, checkpoint software package so that the RTI team, as well as the UW/FH group, could perform appropriate testing and validation of the rapidly evolving software. The RTI team implemented the changes to the TranStat application according to the modification plan and performed initial testing using input data sets provided by the UW/FH group. A separate testing team at RTI performed sequent testing to ensure objective implementation reviews. During development, the UW/FH group also tested and revised interim versions of the TranStat application in order to maintain compatibility between the revised GUI and the underlying computational model.

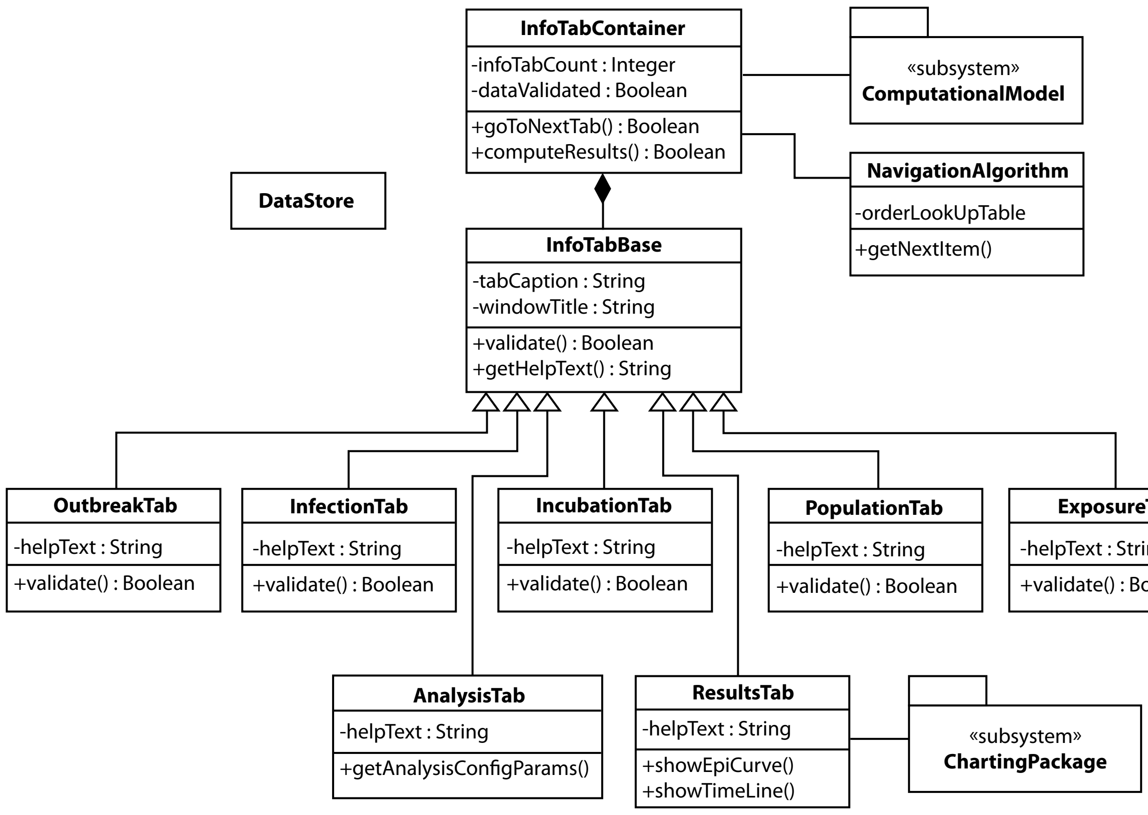
The two teams worked together to slightly modify the software development process to allow greater latitude for incorporating critical modifications discovered during testing and validation. In addition, the RTI team implemented urgent modifications and tested them before proceeding to the next iterative cycle of development.

Features of the Revised TranStat Application

Figure 5 shows key elements of the revised TranStat application using UML notation. Using UML enables programmers to discuss or exchange key elements of software design independent of programming language and platform. The modular design shown in Figure 5 supports the addition of new ordered data entry pages that segment data entry and analysis features into manageable groupings and allows for changes in the order of the pages and the incorporation of other output display libraries. The ability to reuse the application may reduce start-up time for related investigations.

As shown in Figure 5, the revised TranStat application's tabbed display functionality is contained in the InfoTabContainer. Each data input page is presented by a class derived from the InfoTabBase class. The navigation scheme that dictates the order of

Figure 5. Revised TranStat application design



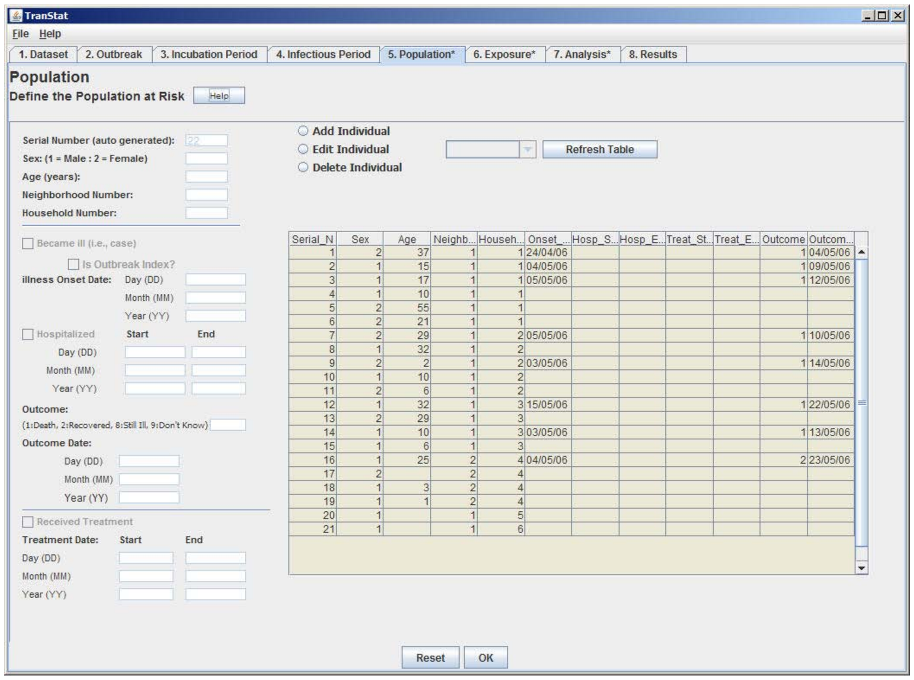
data input is contained in the NavigationAlgorithm class. Data input pages are represented as derived classes, labeled OutbreakTab, InfectionTab, IncubationTab, PopulationTab, and ExposureTab. The derived classes have a validate() method containing instructions for ensuring that collected data meet expected limits, and a helpText variable containing assistive descriptions of the input data. The AnalysisTab class collects parameter values for establishing the analysis constraints. The ResultsTab class displays calculated values of epidemiological metrics and issues commands to the charting library to display analysis results as a graph or chart. The computational model and the charting library are represented in the design as <<subsystem>> ComputationalModel and ChartingPackage modules, consisting of groupings of classes marked as a single functional module.

The revised TranStat application has a modified GUI, a revised validation scheme that ensures that input data are within the boundaries and specifications of the computational model to produce results, and an extended graphical results output. The revised application also includes annotated data entry fields and output graphics with expanded explanatory labels and assistive text display options.

Figure 6 shows the revised TranStat GUI, which allows for dynamic display capabilities, integrated input validation, and directed navigation among data entry screens. As shown, the user experience is enhanced by the ordered data entry pages that segment data entry and analysis features into manageable groupings. The navigation tabs guide a user through the process of using the computation model, including data input, data analysis invocation, and the display of results. The revised user interface also includes a Help option, which displays additional explanatory text. The descriptive labeling, extensive help options, and guided direction of data input assist new users in using the computational model.

The RTI team augmented the data input process to include data validation and extended help features to assist users with specifying input parameters. To support the validation, we introduced a navigation capability to enforce the order of data entry and to

Figure 6. Revised TranStat user interface



Population
Define the Population at Risk

Serial Number (auto generated): 22
Sex: (1 = Male : 2 = Female)
Age (years):
Neighborhood Number:
Household Number:

Became ill (i.e., case)
Is Outbreak Index?
illness Onset Date: Day (DD)
Month (MM)
Year (YY)
Hospitalized Start End
Day (DD)
Month (MM)
Year (YY)
Outcome:
(1:Death, 2:Recovered, 8:Still Ill, 9:Don't Know)
Outcome Date:
Day (DD)
Month (MM)
Year (YY)

Received Treatment
Treatment Date: Start End
Day (DD)
Month (MM)
Year (YY)

Serial_N	Sex	Age	Neighb...	Househ...	Onset_...	Hosp_S...	Hosp_E...	Treat_St...	Treat_E...	Outcome	Outcom...
1	2	37	1	1	24/04/06					1	04/05/06
2	1	15	1	1	04/05/06					1	09/05/06
3	1	17	1	1	05/05/06					1	12/05/06
4	1	10	1	1							
5	2	55	1	1							
6	2	21	1	1							
7	2	29	1	2	05/05/06					1	10/05/06
8	1	32	1	2							
9	2	2	1	2	03/05/06					1	14/05/06
10	1	10	1	2							
11	2	6	1	2							
12	1	32	1	3	15/05/06					1	22/05/06
13	2	29	1	3							
14	1	10	1	3	03/05/06					1	13/05/06
15	1	6	1	3							
16	1	25	2	4	04/05/06					2	23/05/06
17	2		2	4							
18	1	3	2	4							
19	1	1	2	4							
20	1		1	5							
21	1		1	6							

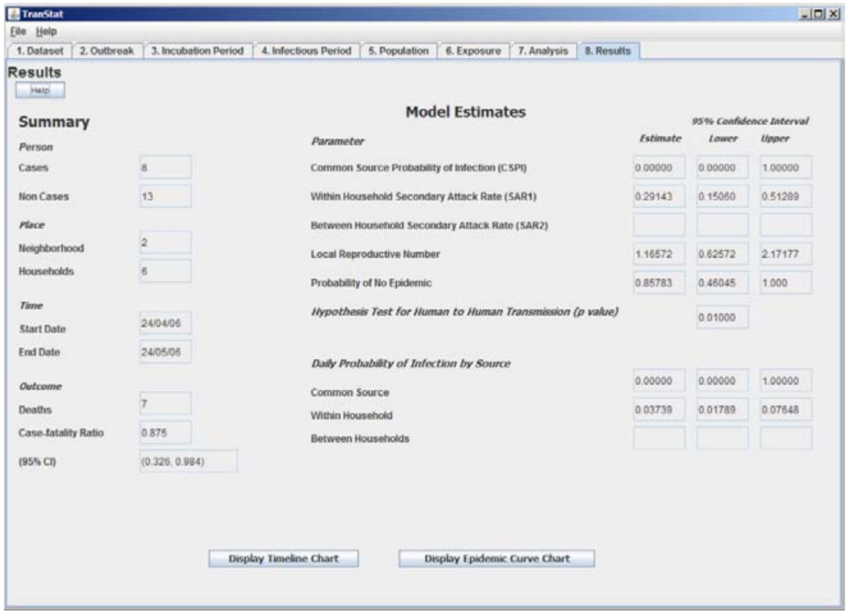
achieve more efficient propagation of changes to data. Data storage and tracking features are updated in turn so that user-initiated adjustments to input parameters are reflected in the system immediately upon validation.

In addition to the modified GUI, the revised application offers two options for displaying analysis output in graph form. In contrast with the initial application output shown in Figure 4, Figures 7–9 show how incorporating the data visualization library into the TranStat application enhanced the presentation of analysis results considerably. In particular, it enabled the presentation of high-quality graphs and charts containing dynamically generated explanatory annotations.

Figure 7 shows the calculated analysis results in a tabular form. Figure 8 shows an example output chart for a projected epidemic curve. For each day of the simulated outbreak, the table in the top left of Figure 8 shows each household, labeled HH, and the number of cases or persons infected. The information in the table is also presented as a graph, the epidemic curve, with each household represented as a distinctly colored bar in that day's column. The epidemic curve further shows the total number of infected cases for each day of the simulated outbreak.

Figure 9 shows the calculated output of the simulated outbreak timeline. For each member of the simulated population, identified by the subject number (SN), the graph shows the duration of their illness in the simulation. The color

Figure 7. TranStat analysis output: tabular information



Results

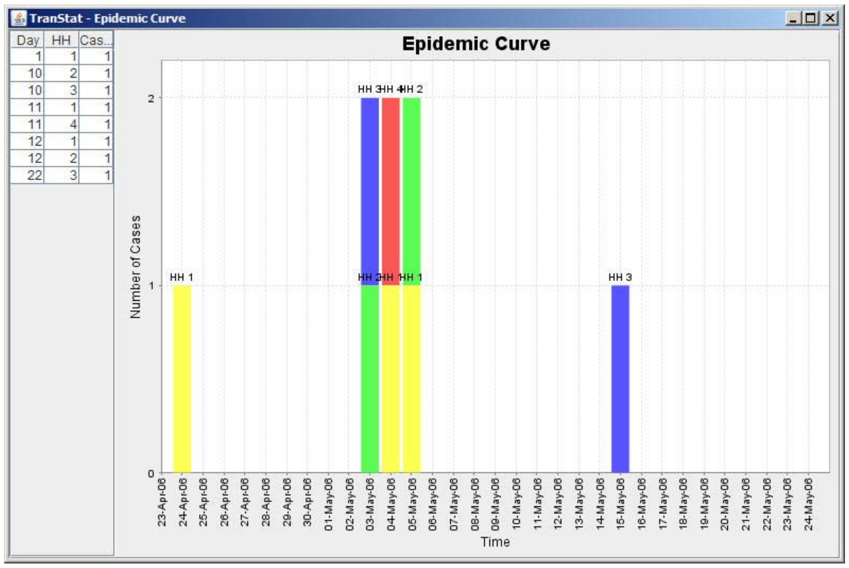
Summary

Person	
Cases	8
Non Cases	13
Place	
Neighborhood	2
Households	6
Time	
Start Date	24/04/06
End Date	24/05/06
Outcome	
Deaths	7
Case-fatality Ratio	0.875
(95% CI)	(0.326, 0.984)

Model Estimates

Parameter	*Estimate*	*95% Confidence Interval Lower*	*95% Confidence Interval Upper*
Common Source Probability of Infection (CSPI)	0.00000	0.00000	1.00000
Within Household Secondary Attack Rate (SAR1)	0.29143	0.15060	0.51289
Between Household Secondary Attack Rate (SAR2)			
Local Reproductive Number	1.16572	0.62572	2.17177
Probability of No Epidemic	0.85783	0.46045	1.000
Hypothesis Test for Human to Human Transmission (p value)		0.01000	
Daily Probability of Infection by Source			
Common Source	0.00000	0.00000	1.00000
Within Household	0.03739	0.01789	0.07648
Between Households			

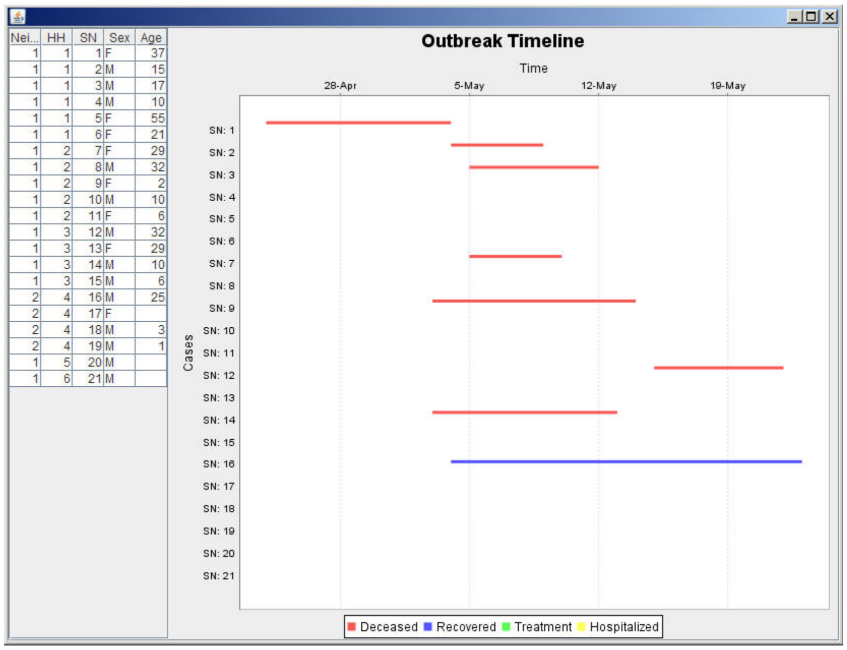
Figure 8. TranStat analysis output: epidemic curve



Day	HH	Cas...
1	1	1
10	2	1
10	3	1
11	1	1
11	4	1
12	1	1
12	2	1
22	3	1

of the line indicates the subject's final disease status (i.e., deceased, recovered, treatment, hospitalized). The accompanying table presents a summary of information for each subject, including his or her assigned neighborhood, household, subject number, sex, and age.

Figure 9. TranStat analysis output: outbreak timeline



Nei...	HH	SN	Sex	Age
1	1	1	F	37
1	1	2	M	15
1	1	3	M	17
1	1	4	M	10
1	1	5	F	55
1	1	6	F	21
1	2	7	F	29
1	2	8	M	32
1	2	9	F	2
1	2	10	M	10
1	2	11	F	6
1	3	12	M	32
1	3	13	F	29
1	3	14	M	10
1	3	15	M	6
2	4	16	M	25
2	4	17	F	
2	4	18	M	3
2	4	19	M	1
1	5	20	M	
1	6	21	M	

Obtaining the TranStat Application

The RTI team continues to offer the distribution of TranStat to the public. The application is available without restriction through the MIDAS portal, https://www.epimodels.org /midas/about.do.[4] The portal offers users the ability to submit comments and raise issues regarding the latest version of TranStat.

Discussion

Computational models are valuable tools in characterizing infectious disease events, especially in situations where access to timely information is crucial for rapid response. Researchers' ability to leverage existing tools can expedite subsequent investigations using stable and robust tools. Moreover, software implementation of computational models developed within research groups may be of interest to other research groups. In its original state, however, the software may not be immediately usable by the other researchers. By using software engineering best practices—such as the ISDP, OOM, and UML—developers can develop software applications that are extensible and easier to maintain, and that support collaboration among researchers.

The resulting tools can provide a common toolset for researchers with common interests, which can facilitate activities such as collaborative discussions and encourage research in new directions.

Whereas the TranStat modification effort was limited to addressing the application's user interface aspects, researchers can incorporate software engineering principles during the design and development of computational models to develop models that other research groups can use or build upon. The ISDP and sufficient oversight can help manage software development while supporting dynamic research discovery. Researchers can use OOM to develop computational models that provide a basis for subsequent work, with application modules that are adaptable to other computational modeling efforts. Finally, computational model designs expressed in UML notation can present the model approach, relationships, and lessons learned in a manner independent of programming language, thereby supporting peer review and understanding.

Although some planning is involved in applying software engineering techniques to the development of computational models, the resulting UML designs and computational models represent a tested and peer-reviewed repository of tools to expedite research investigations.

References

1. National Institute of General Medical Sciences. Models of Infectious Disease Agent Study. [Internet]. [cited 2010 July 29]. Available from: http://www.nigms.nih.gov/Initiatives/MIDAS/.
2. Yang Y, Halloran M, Sugimoto J, Longini I. Detecting human-to-human transmission of avian influenza A (H5N1). Emerg Infect Dis. 2007;13(9):1348-53.
3. National Institute of General Medical Sciences. MIDAS TranStat Fact Sheet. Models of Infectious Disease Agent Study [Internet]. [cited 2010 July 29]. Available from: https://www.epimodels.org/midasdocs/factsheets/MIDAS_TranStat_web.pdf
4. National Institute of General Medical Sciences. Models of Infectious Disease Agent Study [Internet], [cited 2010 July 29]. Available from: https://www.epimodels.org/midas/about.do
5. Larman C, Basili V. Iterative and incremental development: a brief history. Computer. 2003;36(6):47-56.
6. Booch G. Object-oriented analysis and design with applications. 2nd ed. Redwood City (CA): Benjamit/Cumming Publishing; 1994.
7. Quatrani T. Introduction to the unified modeling language. [Internet]. DeveloperWorks: IBM's Resource for Developers' Web site. [Updated 2003 Nov 24; cited 2010 July 29]. Available from: http://www.ibm.com/developerworks/rational/library/998.html
8. Maltz, E. Is all communication created equal?: an investigation into the effects of communication mode on perceived information quality. Journal of Product Innovation Management, 2000:17(2): 110-27.
9. Jigloo SWT/swing GUI builder for Eclipse and WebSphere [Internet]. [cited 2010 Aug 5]. Available from: http://www.cloudgarden.com/jigloo/.
10. JFreeChart [Internet]. [cited 2010 Aug 5]. Available from: http://www.jfree.org/jfreechart/.

Acknowledgments

The authors wish to thank Yang Yang, PhD, affiliated with the Fred Hutchinson Cancer Research Center, Seattle, Washington, and Jonathan Sugimoto and Ira Longini, PhD, both affiliated with the Fred Hutchinson Cancer Research Center and the University of Washington, Seattle, for contributing their computational model implementation to this effort. This work was supported by cooperative agreement U24GM087704 (Models of Infectious Disease Agents Study Information Technology Resource) from the National Institute of General Medical Sciences of the National Institutes of Health.