# SOFTWARE

## CropSyst: A Collection of Object-Oriented Simulation Models of Agricultural Systems

F. K. Van Evert* and G. S. Campbell


### ABSTRACT

**Simulation of whole agricultural systems is now widely used in agronomy. Construction and maintenance of the large simulation models required for agricultural systems may benefit from the application of modern programming methods. In particular, object-oriented programming (OOP) methods claim several advantages over conventional procedural methods. We sought a programming approach that would allow (i) interchanging of component models within and between whole-system models, (ii) incremental model building without rewriting existing code, (iii) maintenance of more than one model of a component, and (iv) construction of a user-friendly interface from which all parameters can be assigned and component models run. Here we report results of an experiment in which we used OOP to construct a cropping system model called CropSyst. An OOP analysis of cropping systems led to the abstraction of component systems (objects) with minimal and well-defined interfaces. Examples of components, or objects, used in CropSyst are Time, Weather, Crop, Soil, Crop residue, Tillage, Erosion, Aphid population, Aphid immigration, Pesticide application, Planting, Crop rotation, and Output. Different versions of CropSyst were implemented and used to simulate production and erosion for cropping systems in eastern Washington, and to simulate yield loss and pesticide dynamics associated with Russian Wheat Aphid infestation. These were constructed from existing objects. Different versions of the Crop object simulated the different crops in a rotation cycle. Parameters were assigned and models were run from a commercially supplied user interface, which was also programmed using OOP. We were able to meet our objectives using OOP, and found it useful for construction and maintenance of agricultural systems models.**


WHOLE-SYSTEM MODELS are typically large, complex, and expensive to create and maintain. The complexity of the model reflects the complexity of the system, and is therefore not something the modeler can avoid. Modeling tools are available, however, to help manage model complexity and minimize expense. The objective of this work was to investigate the use of object-oriented programming (OOP) as a tool for creation and management of whole-system models for agricultural applications. Specifically, we sought a programming approach that would allow (i) interchanging of component models within and between whole-system models, (ii) incremental model building without rewriting existing code, (iii) maintenance of more than one model of a component, and (iv) construction of a user-friendly interface from which all parameters can be assigned and component models run. These goals are not new to modelers, and are met to some extent in existing models using conventional programming languages (Hodges et al., 1992; Buttler and Riha, 1989). However, an object-oriented approach (Booch, 1991; Wegner, 1990; Cox, 1986; Rossiter, 1991) seemed to provide the tools to meet all of these objectives, and to provide a model structure that in some ways was similar to the system being modeled. We report here some of the results of experiments with OOP in construction of a cropping systems model called CropSyst. While the components of this version of CropSyst are available and easy to use, they are intended primarily for use by modelers. A closely related effort is directed at nonmodelers (Stockle et al., 1991).

### OBJECT-ORIENTED ANALYSIS AND DESIGN

The goal of object-oriented analysis and design is to model a real-world system using objects that are abstracted from the problem domain. As such, it is different from structured analysis and design, which focuses on data flows and algorithms (Yourdon, 1989; DeMarco, 1979). An object is an identifiable item, either real or abstract, with a well-defined role and boundary or interface (Smith and

F.K. Van Evert, ICRISAT Sahelian Center, B.P. 12404, Niamey, Niger (via Paris); and G.S. Campbell, Dep. of Crop and Soil Sciences, Washington State Univ., Pullman, WA 99164-6420. Contribution of the Washington Agric. Exp. Stn. Paper no. 9201-51. Received 24 July 1993. *Corresponding author.

Published in Agron. J. 86:325-331 (1994).

**Abbreviations:** IC, integrated circuit; EMS, expanded memory; K, kilobyte; MB, megabyte; OOP, object-oriented programming.

Tockey, 1988; Cox, 1986). A useful analog is an integrated circuit (IC) used in electronic design. The IC has a specific function and interface. The circuit designer knows, from published information, how to connect the IC and what its output is for any specified input. This same IC can be used in many different circuit designs without the designer ever needing to know how the IC was designed. Likewise, an object contains data and code that allow it to function in a specified manner. The interface is provided by the object's methods (called procedures, functions, or subroutines in procedural languages). Since the interface and the function of an object are defined, the object can be used in numerous applications without rewriting the code, and the user need not be capable of producing the object in order to use it.

While some features of OOP can be emulated in procedural languages, those languages which are designed for OOP implement three unique features that are particularly useful in the construction and maintenance of agricultural systems models. These are *encapsulation, inheritance,* and *polymorphism* (Borland, 1990).

Encapsulation combines data and the computer code that generates or manipulates that data into a single unit called an object. Access to both the code and the data is through the object's methods. The user therefore never needs to know whether a requested piece of information was computed within the object or obtained from stored records. As an example of the usefulness of this feature, two weather objects might be maintained, one that supplies daily temperature and precipitation data from a historical file, and one that generates data using statistical routines. For other objects using this information, the source of the data is unimportant. As long as the interface is the same for both weather objects, either object can be used interchangeably with other objects. Changes and improvements in the weather algorithms within the object can also be made, but other objects will not be affected, so long as the interface remains unchanged.

Inheritance allows the extension of object capabilities without rewriting code. Descendent objects can inherit methods and data from ancestor objects simply by specifying the name of the ancestor in the descendent object's code. In the descendent object, new methods can be added, but, more importantly, methods in the ancestor object can be redefined. As an example, we might have a weather object with a method for computing potential evaporation using a simple temperature-based method. A descendent object might alter this method to compute evaporation using solar radiation. The other methods in the weather object that computed temperature or vapor pressure would not have to be included in the new code, because these would be inherited from the ancestor object.

Polymorphism allows two or more objects to share the same interface definition, making it possible for the user to replace one with another when the program runs, thus achieving different results. To illustrate this feature, assume that a model was developed with a single weather object that reads daily weather data from a file and provides CropSyst with the temperature, evaporation, and vapor pressure information required to run the model. Later, another weather object is developed, perhaps by a different person, that simulates weather variables. Polymorphism allows the user to replace the old weather object with the new one, without having access to the code for the old object.

**Table 1. Examples of CropSyst objects with brief descriptions of their function. The one-to-one correspondence of CropSyst's objects to components in the real world is emphasized.**

| Object | Function |
|---|---|
| Time | Keep track of time; supply other component models with information about the current time. |
| Weather | Provide other component models with values for meteorological variables. |
| Crop planting | If the current crop has not yet been planted: monitor the environment and signal that the crop should be planted when the time has come. |
| Crop | Simulate the crop growth and development processes appropriate for the current stage of development. |
| Crop rotation | When the current crop has finished its life cycle: replace current crop, crop planting and tillage system objects with their successors in the crop rotation. |
| Crop residue | Simulate decomposition of crop residue; redistribution of surface and shallowly buried residue when a tillage operation is performed; interception of precipitation, as well as evaporation of water held. |
| Soil | Soil water balance processes (surface runoff, infiltration, surface evaporation, deep percolation); adjust the value of the variable describing the roughness of the soil surface when a tillage operation is performed; pesticide dynamics (degradation, transport, uptake). Field operation. |
| Tillage system | If a tillage operation is scheduled for the current day: make information about this operation available to the other component models. |
| Soil erosion | Calculate daily values of the *C*-factor of the Universal Soil Loss Equation and sum these to give an annual *C*-factor. |
| Aphid population | Simulate growth and development of aphids. |
| Aphid immigration | Provide the aphid population model with the number of aphids immigrating per day. |
| Pesticide application | Apply pesticide when the conditions for application are met. |
| Output | Collate state information from the various component models; do disk and/or screen output. |

Inheritance and polymorphism are particularly useful for a cropping systems model. The common features of crops can be placed in an ancestor crop object, and then inherited by specific descendent crop objects that simulate the individual characteristics of the species and varieties in the cropping system. The appropriate crop object is then used at the appropriate time in a rotation or sequence as the management sequence unfolds in the simulation. Following is a description of CropSyst's objects and specifications, along with some discussion of how the features of OOP helped to implement a cropping systems model.

## PROGRAM DESCRIPTION

### Simulation Models

Table 1 lists CropSyst's components or objects and their function. Four simulation models are currently implemented in CropSyst using these objects. The first one simulates crop growth during one life cycle and has the com-

ponents weather, crop, crop planting, and soil. It is primarily useful for testing modified crop or soil objects without the interaction of other components. The second model is an expansion of the first one and has a crop rotation component. The third simulation model has the components weather, crop, crop planting, crop rotation, soil, soil erosion, soil tillage, and crop residue. We have used this model to calculate crop yield and soil erosion in eastern Washington as a function of climate, tillage system, and crop rotation (Van Evert, 1992). The last model has components weather, crop, crop planting, soil, aphid population dynamics, aphid immigration, and insecticide application. It has been used to simulate the time course of Russian wheat aphid infestation of small grains and the resulting reduction in grain yield (Van Evert, 1992).

## Time

Time is the driving variable of simulation models. In CropSyst, time is represented as the number of days since a fixed date. The integer variable that stores this number is not visible to the user, but the time object provides methods to obtain information about the current time, either as calendar day or month, day, year. It also contains a method to advance the time. Weather objects (see below) directly use the integer representation of time to index the appropriate record of weather data. With time expressed in this unique way, a daily time-step simulation model can be written as a REPEAT . . . UNTIL TIME >= LASTDAY loop, regardless of whether one growing season is to be simulated (200 loop passes), or more than 100 yr (40 000 loop passes) for stochastic simulation.

## Weather

It is the task of weather objects to provide information about the current day's weather. Weather data are stored by year in ASCII files or generated stochastically. The data for one year are held in memory at a time. At the beginning of each simulated day, that day's number is passed to the weather object. It checks to see whether the data for that day are in memory and, if not, loads the appropriate file or generates the data for the new year. Then the values of derived weather parameters are calculated for that day and temporarily stored.

We implemented three weather objects. The first requires daily inputs of precipitation (mm) and maximum and minimum temperature (°C). The average daytime vapor density deficit (g $m^{-3}$) is calculated from the difference between maximum and minimum daily temperature and the slope of the saturated vapor density curve at the average temperature (Campbell and Diaz, 1988). Solar radiation (MJ $m^{-2}$ $d^{-1}$) is calculated from the difference in daily maximum and minimum temperature (Bristow and Campbell, 1984). Potential evapotranspiration (kg $m^{-2}$ $d^{-1}$) is calculated with the Priestley–Taylor equation (Priestley and Taylor, 1972). The second weather object differs from the first only in that it requires inputs in Imperial units: i.e., precipitation in inches and temperatures in Fahrenheit. This object is included because some of our weather data are recorded in metric, others in Imperial units. The third weather object implements a weather generator that generates precipitation and daily minimum and maximum temperatures. The weather generator generally follows Richardson and Wright (1984), but some improvements have been made (Van Evert, 1992). Derived weather parameters are calculated as described for the other two weather classes.

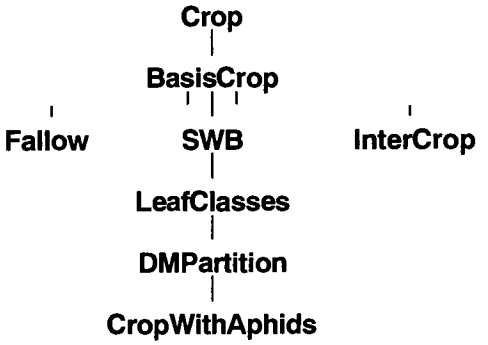


**Fig. 1. Diagram of crop objects in CropSyst illustrating *inheritance* in object-oriented programming. Ancestors and descendants of each crop object are shown.**

## Crop

The crop object hierarchy currently is the most extensive one in CropSyst (Fig. 1). We will describe this hierarchy in some detail as an illustration of the use of inheritance to derive detailed models from simpler ones without having to rewrite code that does not change from one model to another (Reynolds et al., 1989). At the top of the hierarchy stands an object, CROP, whose only function it is to provide an interface to crop objects without making any assumptions about how descendants might be implemented. Its descendant BASISCROP implements the interception of precipitation by the canopy. To this end it needs to know the leaf area index, but an implementation to calculate leaf area index is not provided. The BASISCROP object has three descendants. The FALLOW object always reports a leaf area index of zero and implements a life cycle (as it were) that ends after a certain number of days. The SWB object is a fully functional model of crop growth based on that of Campbell and Diaz (1988) and extended by Van Evert (1992). It is particularly suitable for the simulation of spring grain crops. SWB reports the current leaf area index of the simulated crop. The INTERCROP object is a model of two or more crops growing simultaneously. These might be a commercial crop and one or more weeds; two or more commercial crops; or several commercial crops and one or more weeds. INTERCROP contains two or more descendants of CROP and implements its behavior by calling on these objects. To report its leaf area index, for example, INTERCROP will query its crop objects about their leaf area index and return the sum of these numbers. To grow, it will calculate the amount of light intercepted by each of the crop objects (Spitters, 1989) and call the Grow method of each object with the light interception as one of the parameters.

More detailed crop growth models were derived from SWB by redefining some of its behavior. The LEAFCLASSES object improves on the simulation of leaf area dynamics by including leaf age classes (Van Keulen and Seligman, 1987). The DMPARTITION object introduces a dynamic scheme of dry matter partitioning as a function of devel-

opment stage, growth rate and level of water stress (Van Evert, 1992). As a descendant of the LEAFCLASSES object, DMPARTITION retains the leaf age classes. Finally, the CROPWITHAPHIDS object models crop damage resulting from aphid feeding by reducing the green leaf area index and by increasing the resistance to water flow in the plant when aphids are present (Van Evert, 1992).

### Crop Planting

Two crop-planting models are provided. The first one plants the crop on a fixed date and can be used to simulate an experiment where the planting date is known. The second planting model is intended to simulate a planting decision in the spring, when the soil must be sufficiently dry to allow planting equipment to enter the field, while temperatures must be high enough to ensure rapid germination. This model calculates the 5-d moving averages of precipitation and average air temperature, beginning on 1 April. The crop is planted as soon as the moving average precipitation is zero and the moving average temperature is $\geq 5.6°C$ (Yan, 1989).

### Soil

Two soil models are provided in CropSyst. In both, the soil is arbitrarily divided into a number of horizontal layers with volumetric water content as the state variable. Both models assume constant physical properties in the entire profile, though layer-specific properties could easily be accommodated if field data were available to describe them. Infiltration of water is modeled with a cascading layer concept (Campbell and Diaz, 1988), where the top soil layer is wet to field capacity, the surplus of infiltration water wets the next layer, and so on, until all water is used or all layers have been wet to field capacity. In the latter case, the remaining water becomes unavailable for use by crops through deep drainage.

Surface runoff is calculated with the curve number method (SCS, 1972) as implemented by Campbell and Diaz (1988). Soil water evaporation is assumed to occur only from the top layer, and depends on the fraction of radiation that is not intercepted by crop or crop residue (Ritchie, 1972) and the wetness of the soil surface (Campbell and Diaz, 1988). Crop water uptake from each soil layer is calculated by the crop object. This information is passed to the soil object, which updates the water contents of each layer. The soil object also keeps track of the roughness of the surface (a factor in the calculation of soil erosion) as affected by tillage and wintering (D.K. McCool, Washington State University, personal communication, 1992).

The second soil model accounts for pesticide dynamics using algorithms adapted from the PRZM model (Carsel et al., 1985). Degradation is modeled as a first-order rate process; transport is modeled as a convective process with numerical dispersion, and uptake by plant roots is assumed to be passive.

### Soil Erosion

The soil erosion model calculates the *C*-factor of the Universal Soil Loss Equation. The *C*- or cover-management factor is defined as the ratio of soil erosion from a particular field to erosion from a clean-tilled, continuously fallow field. The *C*-factor is calculated with an algorithm developed by D.K. McCool (Washington State University, personal communication, 1992) and described by Yan (1989) for Pacific Northwest conditions.

### Crop Residue

Two pools of crop residue that are effective in protecting the soil are recognized: surface residue and shallowly (0 to 10 cm) buried residue. The rate of decomposition of residue depends on its water content, temperature, and a time constant (Bristow et al., 1986; Stroo et al., 1989). Residue may be moved from the surface pool to the shallow pool, and out of the shallow pool, when a tillage operation is carried out. The fraction of surface residue buried and the fraction of shallow residue lost depend on the tillage operation and are obtained from the soil tillage model.

The crop residue on the surface intercepts a fraction of the solar radiation incident on it and thus reduces evaporation from the soil surface. It also intercepts a fraction of the precipitation and loses this water again via evaporation. If the residue becomes saturated with water, any subsequent intercepted precipitation leaches onto the soil surface.

### Field Operation

Field operations include any mechanized operations. Field operations may redistribute crop residue (see Crop residue). They also may affect the roughness of the soil surface. A field operation object's behavior consists of reporting, to the soil tillage object, the change in surface roughness and the fractions of residue to be moved.

### Soil Tillage System

The tillage model compares the day and month of the current date to the list of dates and field operations it has, and, if it is found that an operation must be performed today, it makes the following information available to the other component models: the fraction of crop residue on the soil surface to be buried in the top 10 cm of soil, and the fraction of residue in the top 10 cm of soil to be buried deeper (*C*-factor software, Soil Conservation Service, Spokane, WA, personal communication, 1989; Yan, 1989).

### Crop Rotation

When the current crop has finished its life cycle, a crop rotation object replaces the current crop object with an object representing the next crop in the rotation. At the same time, other objects such as the tillage system or the planting object may need to be replaced. There are three crop rotation models. The first one replaces just the crop and crop planting objects and is used in a weather–crop–soil simulation. The second one replaces the tillage object in addition to the crop and crop planting objects. The third rotation model implements a flexible rotation (Young and Van Kooten, 1989). It replaces a spring crop at the end of its life cycle with a fallow object and then may replace

the fallow object with another spring crop on a fixed date in the spring. This second replacement takes place only if the amount of plant-available water in the soil profile is larger than a predetermined amount (Young and Van Kooten, 1989).

### Aphid Population

The model of aphid population dynamics is based on that of Carter et al. (1982) for the English grain aphid [*Sitobion avenae* (Fabricius)]. We retained most of the structure of that model, but excluded predation and diseases. Processes included are immigration, development, reproduction and survival. The time step for development and survival is 1 h, the same as in the original model; for immigration and reproduction, we used 1 d. We derived specific relationships for the Russian wheat aphid [*Diuraphis noxia* (Mordvilko)] from recent literature and simulated yield reduction of small grains resulting from infestation by this aphid (Van Evert, 1992).

### Aphid Immigration

The aphid population of the model described above is initialized when aphids immigrate. Our immigration model calculates immigration from observed flight data.

### Pesticide Application

We implemented only one pesticide application model. It applies a predetermined amount of pesticide when aphid infestation reaches a predetermined threshold.

### Output

Although not a component of agricultural systems, we have included an output object in our simulation models. This object collects and collates state information from all the other objects. Several output classes are available to provide different run-time graphical screens, as well as file output for later study. One output object provides no graphical output and is used if simulations are run in batch mode. The state variables to be stored or plotted are selected when a simulation is set up using the CropSyst shell (see below).

## Inter-Object Communication in CropSyst

While the bulk of information flow goes on inside the objects, some exchange takes place between objects. Output from an object is obtained through calls to its methods. Inputs are passed once every time step via a record variable. All of CropSyst's classes have a method that makes the calculations for 1 d. While all descendants of, for example, CROP, will have this method, their input needs may be different. We use records to meet these different needs. Each object accesses only those fields of the input record that are needed. For each component system an input record type is defined. For example, some of the fields of the input record for the crop component are average air temperature, precipitation, soil water content, and aphid population density. All crop models use the first

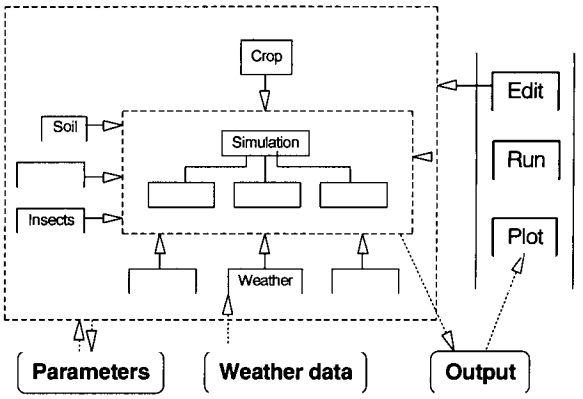


**Fig. 2. Schematic representation of CropSyst. Edit, Run, and Plot are the main functions of the shell. The Edit function accesses any of the component models (denoted by rectangular, thin-line boxes); the component models access and modify the information in the Parameters database (databases are denoted by thin-line, round-edged boxes). The Run function accesses the simulation models; the dashed-line box encloses the simulation object hierarchy. Arrows denote that an object is part of another object: simulation models may contain crop, soil, weather and several unnamed models. The Weather object accesses a database with weather data. When a simulation model is executed, its output is stored in an output database. The Plot option accesses this database to generate reports.**

three inputs, but only the crop model that implements crop response to the presence of aphids uses the last input.

A special case is the exchange of information about soil water content, soil water potential, and water uptake from the soil. This information is in vector form, with one element for each soil layer. In order to increase efficiency, we declare a global type SOILVECTOR and transfer pointers to arrays rather than the arrays themselves.

## CropSyst's Shell

The main functions of the CropSyst shell are (i) to edit an object (i.e., a component or simulation model), (ii) to run a simulation model, and (iii) to store or plot the results of a simulation on the computer screen. These functions are invoked from a shell (Fig. 2).

Editing an object involves retrieving, viewing, and possibly modifying and storing the values of its parameters and the initial values of its state variables (a parameter set). Simulation models and component models can both be edited. The edit process for a crop model is illustrated in Fig. 3. The parameters for a simulation model are the first and last day of the simulation, plus for each component an identification of the model to be used and the parameter set to be used with that model.

In order to run a simulation model, the user must select the simulation model and the parameter set to be used. Both choices are made by picking from a list with names. State variables to be stored during the simulation are also specified. Once a model has been executed, output that has been stored on disk can be retrieved and plotted on the screen with the Plot option. This option has currently only been implemented in a primitive manner.

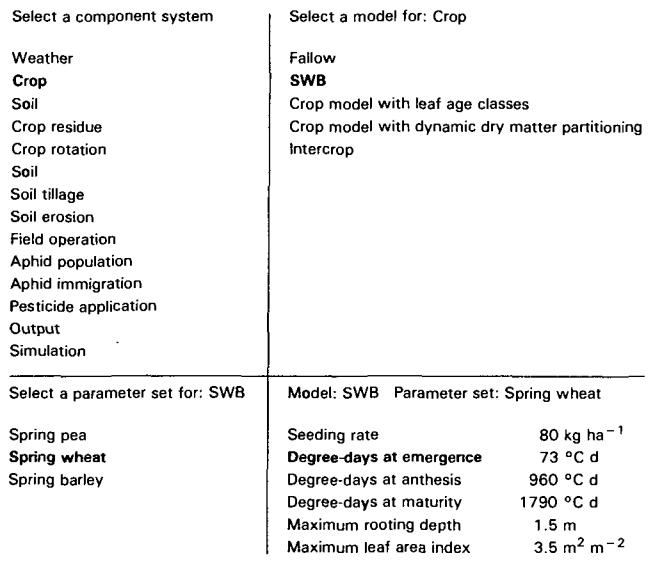


**Fig. 3. The steps in editing a component model. Top left: A view of the computer screen after the Edit option from the main menu has been selected; now the type of component model to be edited can be entered (Crop is currently highlighted). Top right: The crop model type has been selected and a list of available crop models is shown. Bottom left: The user has chosen the SWB model; a list of parameter sets available for this model is displayed. Bottom right: One of several screens on which parameter values for the Spring Wheat parameter set of the SWB model are shown. These values can now be altered.**

## SPECIFICATIONS

### Software, Hardware, and Performance

The version of CropSyst described here was written in Turbo Pascal, version 6.0 (Borland International, Scotts Valley, CA), run under DOS 5.0. The Turbo Vision application framework, which is included with Version 6 of Turbo Pascal, was used extensively to implement the shell. Turbo Vision provides facilities for editing, reading from and writing to files, displaying windows, and responding to mouse commands. A database management system was therefore readily available for managing parameter sets.

The source code for CropSyst occupies 500K. Additional code from the Turbo Vision system is also used. The size of the executable code is 250K, of which 150K is overlaid. An IBM-compatible computer with 1 MB of memory, DOS 3.3 or above, a hard-disk, and any monitor is required to run the program. Much better performance is obtained if a numerical coprocessor is available, DOS 5.0 is used to free up low memory, and 0.5 MB or more of expanded memory (EMS) is available to store the overlays. Component model parameters and weather data (several hundred kilobytes, depending on use) are stored on disk. A mouse may be used to make menu selections. A 1-yr simulation of a system with components weather, crop, crop planting, crop rotation, crop residue, soil, soil tillage, soil erosion, and graphical screen output takes <10 s on a 486-33MHz with DOS 5.0, 1 MB conventional memory and 1 MB EMS.

### Documentation

Details on implementation and use of OOP and Turbo Vision are provided by Borland (1990). Details on the algorithms used in CropSyst are described in Campbell and Diaz (1988), Yan (1989) and Van Evert (1992). Some informal technical documentation on CropSyst's objects and the implementation of the systems models described is available directly from the corresponding author.

### Software Availability

Contact the corresponding author for a free copy of the source code, a compiled version of CropSyst, and a file with technical documentation.

## DISCUSSION AND CONCLUSIONS

Our analysis of an agricultural system led to the abstraction of object classes that correspond to simple subsystems in the real-world system. Using object-oriented programming, we were able to create components or objects that simulated the behavior of these subsystems. These were used in three different systems models. Several versions were created for some objects, and these were interchanged, depending on the requirements of a specific model.

Object-orientation facilitated the stepwise development of CropSyst. Component systems could be modeled and programmed separately in classes, which were later joined via clear and limited interfaces. This means that, instead of having to deal with the whole system at once, we could start by implementing a small part of the system. For example, at one time we had an aphid population model without any other models around it. Inputs needed by the aphid model were provided by supplying constant values to the aphid model interface. Only after we had gained confidence in the aphid model performance did we include other models to calculate the input values required by the aphid model.

Similar results can be obtained through a carefully designed procedural analysis of an agricultural system (Hodges et al., 1992; Buttler and Riha, 1989). However, procedures do not allow the state of components to be specified together with the code that acts on them. In order to transfer or replace a component one would not only have to replace the procedures, but also to identify all of the variables defining the state of the component and replace them as well. This can be very difficult in large models.

Inheritance allowed us to meet our second objective, the development of new (component) models without rewriting existing code. Because a descendant model can appear in place of one of its ancestors, it was possible to specify that a simulation model contains, for example, a crop component model and to decide at run-time (after compilation) which descendant crop model to use for a particular run.

We benefited extensively from inheritance in writing the shell program. Having all models descend from a common ancestor made it possible to write code for parameter inspection, editing, storage, and retrieval only once and to pass this functionality on to all descendants. In the process of developing a model, code is written and rewritten many times. During that time, the number and type of parameters used by the model may change. Because the base object takes care of parameter input, checking, storage, and retrieval, the shell was used to create and maintain parameter sets for the new model at all stages of de-

velopment. This turned out to be an excellent tool for maintaining an orderly organization of parameter values during the sometimes very disorderly process of model development.

An interesting result of our object-oriented analysis of agricultural systems was the narrowing of the definition of a crop model to comprise only crop processes such as growth and development. This excluded some processes that are often implicitly assumed to be part of crop models, namely weather, soil, and the crop planting decision (e.g., Spitters et al., 1989). As a result, CropSyst's component models should be more portable and more easily used in whole-system models that use different soil, weather, or crop planting models.

The degree of modularity made possible by object-orientation may well make it possible to share models between developers more effectively than has previously been possible. In our experience, it has been easier to reacquaint ourselves with our own object-oriented code than with our own procedure-oriented code. From this observation, we deduce that it will be easier to share component models between individuals in object-oriented form than in procedure-oriented form. However, in order to efficiently exchange object-oriented models it will be necessary to develop interface standards.

## ACKNOWLEDGMENTS

Valuable comments on an early version of the manuscript were received from Drs. D.J. Mulla and C.O. Stockle. The associate editor and reviewers also made many helpful comments. Partial support for this work was received from a USDA STEEP II grant.

## REFERENCES

Booch, G. 1991. Object-oriented design with applications. Benjamin Cummings, Redwood City, CA.

Borland International. 1990. Turbo Pascal 6.0 user's guide. Borland Int., Scotts Valley, CA.

Bristow, K.L., and G.S. Campbell. 1984. On the relationship between incoming solar radiation and daily maximum and minimum temperature. Agric. For. Meteorol. 31:159–166.

Bristow, K.L., G.S. Campbell, R.I. Papendick, and L.F. Elliott. 1986. Simulation of heat and moisture transfer through a surface residue–soil system. Agric. For. Meteorol. 36:193–214.

Buttler, I.W., and S.J. Riha. 1989. General purpose simulation model of water flow in the soil–plant–atmosphere continuum. Appl. Agric. Res. 2:230–234.

Campbell, G.S., and R. Diaz. 1988. Simplified soil-water balance models to predict crop transpiration. p. 15–26. *In* F.R. Bidinger and C. Johansen (ed.) Drought research priorities for the dryland tropics. ICRISAT, Patancheru, AP, India.

Carsel, R.F., L.A. Mulkey, and L.B. Baskin. 1985. The pesticide root zone model (PRZM): A procedure for evaluating pesticide leaching threats to groundwater. Ecol. Modeling 30:49–69.

Carter, N., A.F.G. Dixon, and R. Rabbinge. 1982. Cereal aphid populations: Biology, simulation and prediction. PUDOC, Wageningen.

Cox, B.J. 1986. Object-oriented programming: An evolutionary approach. Addison-Wesley, Reading, MA.

DeMarco, T. 1979. Structured analysis and system specification. Prentice-Hall, Englewood Cliffs, NJ.

Hodges, T., S.L. Johnson, and B.S. Johnson. 1992. A modular structure for crop simulation models: Implemented in the SIMPOTATO model. Agron. J. 84:911–915.

Priestley, C.H.B., and R.J. Taylor. 1972. On the assessment of surface heat flux and evaporation using large scale parameters. Mon. Weath. Rev. 100:81–92.

Reynolds, J.F., B. Acock, R.L. Dougherty, and J.D. Tenhunen. 1989. A modular structure for plant growth models. p. 123–134. *In* J.S. Pereira and J.J. Landsberg (ed.) Biomass production by fast-growing trees. Kluwer, Amsterdam.

Richardson, C.W., and D.A. Wright. 1984. WGEN: A model for generating daily weather variables. USDA-ARS, ARS-8.

Ritchie, J.T. 1972. Model for predicting evaporation from a row crop with incomplete cover. Water Resour. Res. 8:1204–1213.

Rossiter, D.G. 1991. Modern computer programming techniques for environmental simulation modeling. Cornell Univ. Dep. of Crop, Soil and Atmos. Sci. Teaching Ser. 1. Cornell Univ., Ithaca, NY.

Smith, M., and S. Tockey. 1988. An integrated approach to software requirements definition using objects. Boeing Commercial Airplane Support Div., Seattle, WA. *Cited p. 82 in* G.J. Booch. 1991. Object-oriented design with applications. Benjamin Cummings, Redwood City, CA.

Soil Conservation Service. 1972. National engineering handbook. USDA-SCS, Washington, DC.

Spitters, C.J.T. 1989. Weeds: population dynamics, germination and competition. p. 182–216. *In* R. Rabbinge et al. (ed.) Simulation and systems management in crop protection. PUDOC, Wageningen.

Spitters, C.J.T., H. Van Keulen, and D.W.G. Van Kraalingen. 1989. A simple and universal crop growth simulator: SUCROS87. p. 147–181. *In* R. Rabbinge et al. (ed.) Simulation and systems management in crop protection. PUDOC, Wageningen.

Stockle, C.O., F.K. Van Evert, R. Nelson, and G.S. Campbell. 1991. CropSyst: A cropping systems simulation model with GIS link. p. 78. *In* Agronomy abstracts. ASA, Madison, WI.

Stroo, H.F., K.L. Bristow, L.F. Elliott, R.I. Papendick, and G.S. Campbell. 1989. Predicting rates of wheat residue decomposition. Soil Sci. Soc. Am. J. 35:91–99.

Van Evert, F.K. 1992. Modeling agricultural systems with CropSyst. Ph.D. diss. Washington State Univ., Pullman (Diss. Abstr. 93-21059).

Van Keulen, H., and N.G. Seligman. 1987. Simulation of water use, nitrogen nutrition and growth of a spring wheat crop. PUDOC, Wageningen.

Wegner, P. 1990. Concepts and paradigms of object-oriented programming. OOPS Messenger 1:7–87.

Yan, Y., 1989. A model for predicting soil loss ratios and crop production in Eastern Washington. M.Sc. thesis, Washington State Univ. Pullman.

Young, D.L., and G.C. Van Kooten. 1989. Economics of flexible spring cropping in a summer fallow region. J. Prod. Agric. 2:173–178.

Yourdon, E. 1989. Modern structured analysis. Prentice-Hall, Englewood Cliffs, NJ.