DISCRETE EVENT SIMULATION

INTRO AND EXAMPLE CASE IN PROJECT MANAGEMENT

This text is a part of the following publication:

Download the full version (includes SD and ABM)
**Discrete event simulation (DES)** is almost as old as system dynamics. In October 1961, IBM engineer Geoffrey Gordon introduced the first version of GPSS (General Purpose Simulation System, originally Gordon’s Programmable Simulation System), which is considered to be the first method of software implementation of discrete event modeling (Borshchev 2013).

These days, discrete event modeling is supported by many software tools. From a practitioner’s point of view, we should comment that when people apply simulation, the model is a discrete event or includes a discrete event subpart in over 50% of cases.

The main idea of DES is to consider the system as a process, that is a sequence of operations being performed across entities. The operations include delays, service by various resources, choosing the process branch, splitting, combining, and so on. As long as entities compete for resources and can be delayed, queues are present in virtually any discrete event model.

The model is specified graphically as a process flowchart, where blocks represent operations. The flowchart usually begins with “source” blocks that generate entities and inject them into the process and ends with “sink” blocks that remove entities from the model. This type of diagram is familiar to the business world as a process flowchart and is ubiquitous in describing their process steps.

All major DES tools support the same set of basic flowchart blocks such as source, queue, delay/process/service, and sink. There are more than 50 different DES tools, some of them are general purpose such as Arena, AnyLogic, Simul8, and ExtendSim; some are industry-specific, for example, Siemens Plant Simulation, FlexSim, and Automod. You can read the paper “Software Comparison” for more details about different DES products.

A general-purpose software tool can be easily applied for solving project management challenges using DES.

Let’s think about how we can create a model of our software project management using DES.

First of all, we need a source for our requirements and sink for developed software. As soon as we have requirements, we need to seize a developer from experienced_staff. Then, we process a piece of the requirements and release a developer (Figure 1).

Let’s feed the model with data. Requirements_source should produce 500,000 lines of code, each of them being an entity. Let’s introduce a new variable called software_development_rate and assign it the following value:

\[
\text{software_development_rate} = \left(1 - \frac{\text{com_overhead(experienced_staff.size())}}{100}\right) \times \text{productivity}
\]
By doing this, we decrease the software development rate on communication overhead.

We define communication overhead with a table function `com_overhead()`. In AnyLogic, we call functions by function names with parameters in brackets. Such table functions can be a result of real system observations.

Table function includes pairs of arguments and values. It allows you to set up a function when you do not know a formula, the function will be defined and data approximated.

The horizontal axis on Figure 2 represents team size, the vertical axis shows overhead percentage. When we have less than five people, we have no overhead on their communication; but if we start with five and later on the number grows — the more people are working on the project — the bigger overhead they generate.

We should set up the process time as `1/software_development_rate`, because this is a time in days that one developer needs for one line of code. We have a team of ten developers so that our process can process up to ten lines of code simultaneously.

Running the model, we get 106 days for project completion time (Figure 3).

Let’s think about how to implement new developer allocation. We can start by adding a new, initially empty resource pool called `new_project_staff`. We should specify that we can use this pool when we seize developer (Figure 4).

A scripting language is a necessary evil of the professional model-building process that some simulation practitioners like the most and some others struggle with the most. Scripting makes modeling tools very powerful and flexible, but not so easy to create and manage.
In our model, we use Java code expression to set up a unique `software_development_rate` for each line of code processed in the `process` block. This expression is executed each time we seize a developer.

Then, let's modify the Java code that we execute on seizing the developer:

```java
software_development_rate = productivity * 
                        (1 - com_overhead(experienced_staff.size() + new_project_staff.size()) / 100) * 
                        (((Developer)unit).experienced? 1 : 0.8) * 
                        (1 - training_overhead / 100 * new_project_staff.size() / experienced_staff.size());
```

Software development rate is assigned a new value that equals productivity decreased by three multipliers:

- The first one to support communication overhead
- The second one to change productivity in case the seized developer is a new employee
- The last multiplier to support that we have to allocate experienced people time for training.

By this expression, we added `new_project_staff` into communication overhead, supported training overhead calculations, and changed productivity for new developers. Processing speed will be unique or each particular entity representing a line of code, depending on the developer and current team size and structure.

The last and most important part is the allocation process for new developers (Figure 5). As the most important part of the processes, it starts with `Source` block and ends with `Sink` block. New developers leave `staff_allocation` source block with `hiring_rate` per month. As soon as a new developer appears, we increase the capacity of `new_project_staff` resource pool.

After that, new developers enter `training` service block, where they immediately start monthly training. After the `training` ends, we increase `experienced_staff`, adding new resources and decreasing `new_project_staff`.

![FIGURE 4. Seize block settings in AnyLogic software tool](image)

![FIGURE 5. DE process diagram – staff hiring and on-boarding](image)
When we run the model with the same parameters, we get slightly different results caused by the discrete nature of \textit{staff\_allocation}. Project completion time is 88 days, and at the end of the project, we have 15 experienced developers and 3 new people (Figure 6).

The model along with its source code is published in AnyLogic Cloud, and you can find it by searching by the name “\textit{Software Process Dynamics DE}” or by the author’s name Sergey Suslov.

You can use \textit{free AnyLogic PLE} to see the full model internals or even continue the development. DESs are usually used for the engineering part of project management, where managers and engineers try to predict the completion time of every project stage and mitigate the risk of being out of schedule due to technical reasons. In comparison with system dynamics or agent-based modeling, DES is used more on low to middle abstraction levels, where detailed business or technology processes play key roles.