2011

SimML Introduction

Simulation Modeling Language for Kanban

This document introduces the SimML file format for describing software development teams and projects that use Kanban

Focused Objective
6/20/2011

Simulation Modeling Language for Kanban Projects – SimML

Kanban Team / Project Modeling

KanbanSim uses a simple modeling format called "SimML" as an input to the simulation engine technology. This extensible markup language allows a simple model to be built quickly, then refined over-time. The format is in an Extensible Markup Language (XML) specifically designed to model software development teams and the process.

The file format is a simple text file conforming to the rules in the W3C specification (see the document posted at http://www.w3.org/TR/xml/ for reference). XML in its basic description is a nested set of open (e.g. <simulation>) and closing (e.g. </simulation>) tags. Values can be specified within the tags, or as attributes within the opening tag (e.g. <simulation name="my name">value here</simulation>).

The file consists of three sections, which will be covered in detail further in this document –

1. Simulation – the root element for the file format
2. Setup – for defining the team, project and events to model.
3. Execute – for defining the types of simulation to execute.

Simulation Section

The simulation section of a SimML file contains the one root element "simulation" and contains the sub sections: Setup and Execute as defined further in this document. Figure 1 shows the basic structure of a SimML file.

The Simulation element has one attribute, name, which is a user defined name for the simulation file. This name is used for matching result files and for user future reference when examining this file.

```xml
<simulation name="Simple team">
  <execute>
  </execute>
  <setup>
  </setup>
</simulation>
```

Figure 1 - A SimXML file has a simulation element at the root of the document, and an execute and setup section within its opening and closing element tags.

Setup Section

Kanban Column Definitions

Kanban uses columns to represent phases a card or story must traverse to move from backlog to completion. SimML allows any number of columns to be defined. The minimum necessary entries for each column are: a name, a unique id, a low and high bound estimate of cycle time in a column, and a WIP limit. When simulating, each card moving through a column will be allocated a work time between the boundaries chosen, and there will never be more than the defined WIP limit number of cards in a column at one time. Figure 2 shows a typical column definition section. The columns will be ordered from lowest id value to the highest.

```
<columns>
  <column id="1" estimateLowBound="1" estimateHighBound="2" wipLimit="1">Design</column>
  <column id="2" estimateLowBound="2" estimateHighBound="4" wipLimit="2">DB</column>
  <column id="3" estimateLowBound="1" estimateHighBound="5" wipLimit="4">Develop</column>
  <column id="4" estimateLowBound="1" estimateHighBound="2" wipLimit="2">Test</column>
  <column id="5" estimateLowBound="1" estimateHighBound="1" wipLimit="1">DevOps</column>
</columns>
```

Figure 2 - Typical column definition for Kanban simulation

Backlog Definition

For simulation to occur, the number of cards in the initial backlog must be specified. For Kanban simulation, a single entry is used to specify the initial backlog size as show in Figure 3.

```
<backlog type="simple" simpleCount="100" />
```

Figure 3 - Simple Kanban initial backlog size.

Blocking Event definition

During the development process events arise that block progress of a card or story. Any number of these events can be defined in SimML. Blocking events target a column, and block cards for an amount of time between the low and high bound estimate specified, for an occurrence rate between the low and high bound occurrence rate specified. Figure 4 shows a simple definition of two typical blocking events, one for missing requirements found during the development phase, and one for blocking testing due to testing environment issues. Any number of blocking events can be defined, including entire team unavailability for non-productive work-time, like entire day planning and estimation sessions.

```
<blockingEvents>
  <blockingEvent columnId="3" scale="1" occurenceLowBound="5" occurenceHighBound="10"
     estimateLowBound="1" estimateHighBound="3">Block dev (missing requirement)
  </blockingEvent>

  <blockingEvent columnId="4" scale="1" occurenceLowBound="3" occurenceHighBound="5"
      estimateLowBound="1" estimateHighBound="2">Block testing (environment down)
  </blockingEvent>
</blockingEvents>
```

Figure 4 - Typical blocking event definition. There can be any number of events defined.

Defect Definition

During development and testing, defects will be found. Defects can be defined to be found in any column, and those defects start back in any column. The occurrence rate can be specified for each defect, and the cycle time for defects through each column can be customized. For example, simple, moderate and complex defects can be defined separately to model the likely occurrence rate and cycle time for fixing each defect style. If the column estimate overrides are omitted, the column estimates for cycle time will be used. Figure 5 shows the definition of two defect types, the second defect type has custom column cycle times specified.

```
<defects>
  <defect columnId="4" startsInColumnId="2"
     scale="1" occurenceLowBound="2" occurenceHighBound="5">Bug found in Testing ({0})
  </defect>

  <defect columnId="5" startsInColumnId="3"
    scale="1" occurenceLowBound="5" occurenceHighBound="15">Bug found in Dev Ops ({0})
    <column id="2" estimateLowBound="1" estimateHighBound="2" />
    <column id="3" estimateLowBound="1" estimateHighBound="1.5" />
    <column id="4" estimateLowBound="1" estimateHighBound="1" />
  </defect>
</defects>
```

Figure 5 - Defect definition. Any number of defects can be specified to occur between occurrence rates, and each defect can optionally have a custom cycle time for each column.

Added Scope definition

Any scope that is incrementally added to a project once it begins is modeled using Added Scope definitions. The occurrence rate how many cards are introduced into the backlog after a certain number of cards are completed is fully definable in a low-bound to high-bound estimate. Added scope can be used to represent scope creep, production support issues, refactoring, or technical spikes for example. Figure 6 shows two simple added scope events being defined. The special string character {0} is a placeholder for a unique identifier for each scope incident; it is optional and can be omitted.

```
<addedScopes>
  <addedScope scale="1" occurenceLowBound="5" occurenceHighBound="10">
     Scope Creep {0}</addedScope>

  <addedScope scale="1" occurenceLowBound="15" occurenceHighBound="25">
     Refactoring {0}</addedScope>
</addedScopes>
```

Figure 6 - Typical added scopes definition. Any number can be added.

Execute Section

The execute section is for issuing commands to the simulation engine for execution. For Kanban simulation, the following execute sections are valid (currently, more planned) –

1. Visual – single simulation run that returns summary statistics and cumulative flow information.
2. Monte-carlo – multiple simulation runs that return summary statistics and histograms aggregated from multiple simulation runs.

Each of these subsections will be explained later in this document.

The execute section has attribute values that apply across all sub execute actions. LimitIntervalsTo is an integer value that stops simulation after the given number of time intervals. This is to avoid a simulation run that will never end due to mis-configuration of the setup definition of the project. DecimalRounding is the other attribute that trim the number of places after the decimal point in the results file.

```
<execute limitIntervalsTo="365" decimalRounding="3">
  <visual></visual>
  <monteCarlo></monteCarlo>
</execute>
```

Visual simulation

Visual simulation executes a single simulation pass and returns analysis on that execution. By including this element in the execute section, a single pass simulation will occur when this file is executed by the simulation engine. Figure 7 shows the abbreviated XML syntax for including a visual element, and executing a single simulation run.

```
<execute limitIntervalsTo="365" >
  <visual />
</execute>
```

Figure 7 - Adding a visual element within the execute element causes a single run simulation to be executed.

Monte-carlo simulation

Monte carlo simulation performs multiple simulation runs over the defined Kanban model, and returns the aggregate statistics for analysis. To execute a monte-carlo simulation, include a monteCarlo tag within the execute section. The monteCarlo tag takes a single attribute, cycles, which specifies the number of simulation runs to execute. The higher the number of cycles the longer simulation will take to complete, and the more system resources will be needed. Figure 8 show how to specify monte-carlo simulation to be executed, and how to specify the number of simulations runs to execute for that analysis.

```
<execute limitIntervalsTo="365" decimalRounding="3">
  <monteCarlo cycles="250" />
</execute>
```

Figure 8 - Addsing a monte-carlo element will cause a monte-carlo simulation run to execute. Cycles specifies how many simulations runs to perform during analysis.

Conclusion

SimML allows complex analysis of software projects to be performed. The extensible nature of this file format allows future expansion in both descriptions of the team and project structure, and also in the types of analysis undertaken on that setup definition.

[END]

About the Author

Troy Magennis is founder of Focused Objective, a consulting firm that aims to improve software development practices and management through better tools and education. Troy has held positions at VP level for many companies in diverse field from Automotive, Financial, Image Rights Management and Travel.

For feedback, Troy can be contacted at –

Troy.magennis@FocusedObjective.com

For more articles like this, visit use at –

http://www.FocusedObjective.com

Appendix A – Full listing of a sample SimML file.

```
<simulation name="Simple team">
  <execute limitIntervalsTo="365" decimalRounding="3">
    <visual />
    <monteCarlo cycles="250" />
  </execute>

<setup>

    <backlog type="simple" simpleCount="100" />

    <columns>
       <column id="1" estimateLowBound="1" estimateHighBound="2" wipLimit="1">Design</column>
       <column id="2" estimateLowBound="2" estimateHighBound="4" wipLimit="2">DB</column>
       <column id="3" estimateLowBound="1" estimateHighBound="5" wipLimit="4">Develop</column>
       <column id="4" estimateLowBound="1" estimateHighBound="2" wipLimit="2">Test</column>
       <column id="5" estimateLowBound="1" estimateHighBound="1" wipLimit="1">DevOps</column>
    </columns>

       <blockingEvents>
         <blockingEvent columnId="3" scale="1" occurenceLowBound="5" occurenceHighBound="10"
            estimateLowBound="1" estimateHighBound="3">Block dev (missing requirement)
         </blockingEvent>

         <blockingEvent columnId="4" scale="1" occurenceLowBound="3" occurenceHighBound="5"
             estimateLowBound="1" estimateHighBound="2">Block testing (environment down)
         </blockingEvent>
       </blockingEvents>


       <defects>
         <defect columnId="4" startsInColumnId="2"
            scale="1" occurenceLowBound="2" occurenceHighBound="5">Bug found in Testing ({0})
         </defect>

         <defect columnId="5" startsInColumnId="3"
           scale="1" occurenceLowBound="5" occurenceHighBound="15">Bug found in Dev Ops ({0})
           <column id="2" estimateLowBound="1" estimateHighBound="2" />
           <column id="3" estimateLowBound="1" estimateHighBound="1.5" />
           <column id="4" estimateLowBound="1" estimateHighBound="1" />
         </defect>
       </defects>

       <addedScopes>
            <addedScope scale="1" occurenceLowBound="5" occurenceHighBound="10">
                  Scope Creep {0}</addedScope>

            <addedScope scale="1" occurenceLowBound="15" occurenceHighBound="25">
                  Refactoring {0}</addedScope>
       </addedScopes>

  </setup>
</simulation>
```