# Contents

1 Structure 1

2 Installation 2
  2.1 Requirements for Simulink Libraries 2
  2.2 Installation of Simulink Libraries 2

I SimSWE (basic) 3

3 Generators 4
  3.1 RequirementsGenerator 4
    3.1.1 Purpose 4
    3.1.2 Component 4
    3.1.3 Inputs 4
    3.1.4 Parameters 5
    3.1.5 Outputs 6
    3.1.6 Generic Description 7
    3.1.7 Simulink Implementation 8
  3.2 FunctionPointGenerator 9
    3.2.1 Purpose 9
    3.2.2 Component 9
    3.2.3 Inputs 9
    3.2.4 Parameters 9
    3.2.5 Outputs 9
    3.2.6 Generic Description 9

4 Estimators 10
  4.1 COCOMOIIEarlyDesign 10
    4.1.1 Purpose 10
    4.1.2 Component 10
    4.1.3 Inputs 10
    4.1.4 Parameters 10
    4.1.5 Outputs 10
    4.1.6 Generic Description 11
  4.2 COCOMOIIPostArchitecture 12
    4.2.1 Purpose 12
    4.2.2 Component 12
    4.2.3 Inputs 12
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2.4</td>
<td>Parameters</td>
<td>12</td>
</tr>
<tr>
<td>4.2.5</td>
<td>Outputs</td>
<td>12</td>
</tr>
<tr>
<td>4.2.6</td>
<td>Generic Description</td>
<td>13</td>
</tr>
<tr>
<td>4.2.7</td>
<td>Simulink Implementation</td>
<td>13</td>
</tr>
<tr>
<td>4.3</td>
<td>SLOCEstimatorBasedOnFunctionPoints</td>
<td>14</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Purpose</td>
<td>14</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Component</td>
<td>14</td>
</tr>
<tr>
<td>4.3.3</td>
<td>Inputs</td>
<td>14</td>
</tr>
<tr>
<td>4.3.4</td>
<td>Parameters</td>
<td>15</td>
</tr>
<tr>
<td>4.3.5</td>
<td>Outputs</td>
<td>15</td>
</tr>
<tr>
<td>4.3.6</td>
<td>Generic Description</td>
<td>15</td>
</tr>
<tr>
<td>4.4</td>
<td>PerceivedQuality</td>
<td>17</td>
</tr>
<tr>
<td>4.4.1</td>
<td>Purpose</td>
<td>17</td>
</tr>
<tr>
<td>4.4.2</td>
<td>Component</td>
<td>17</td>
</tr>
<tr>
<td>4.4.3</td>
<td>Inputs</td>
<td>17</td>
</tr>
<tr>
<td>4.4.4</td>
<td>Parameters</td>
<td>17</td>
</tr>
<tr>
<td>4.4.5</td>
<td>Outputs</td>
<td>18</td>
</tr>
<tr>
<td>4.4.6</td>
<td>Generic Description</td>
<td>18</td>
</tr>
<tr>
<td>4.4.7</td>
<td>Simulink Implementation</td>
<td>18</td>
</tr>
<tr>
<td>5</td>
<td>Generic Activities</td>
<td>19</td>
</tr>
<tr>
<td>5.1</td>
<td>Work</td>
<td>19</td>
</tr>
<tr>
<td>5.1.1</td>
<td>Purpose</td>
<td>19</td>
</tr>
<tr>
<td>5.1.2</td>
<td>Component</td>
<td>19</td>
</tr>
<tr>
<td>5.1.3</td>
<td>Inputs</td>
<td>19</td>
</tr>
<tr>
<td>5.1.4</td>
<td>Parameters</td>
<td>20</td>
</tr>
<tr>
<td>5.1.5</td>
<td>Outputs</td>
<td>20</td>
</tr>
<tr>
<td>5.1.6</td>
<td>Generic Description</td>
<td>20</td>
</tr>
<tr>
<td>5.1.7</td>
<td>Simulink Implementation</td>
<td>21</td>
</tr>
<tr>
<td>5.2</td>
<td>WorkWithCommunicationOverhead</td>
<td>22</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Purpose</td>
<td>22</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Component</td>
<td>22</td>
</tr>
<tr>
<td>5.2.3</td>
<td>Inputs</td>
<td>22</td>
</tr>
<tr>
<td>5.2.4</td>
<td>Parameters</td>
<td>23</td>
</tr>
<tr>
<td>5.2.5</td>
<td>Outputs</td>
<td>23</td>
</tr>
<tr>
<td>5.2.6</td>
<td>Generic Description</td>
<td>24</td>
</tr>
<tr>
<td>5.2.7</td>
<td>Simulink Implementation</td>
<td>25</td>
</tr>
<tr>
<td>5.3</td>
<td>WorkWithExperience</td>
<td>26</td>
</tr>
<tr>
<td>5.3.1</td>
<td>Purpose</td>
<td>26</td>
</tr>
<tr>
<td>5.3.2</td>
<td>Component</td>
<td>26</td>
</tr>
<tr>
<td>5.3.3</td>
<td>Inputs</td>
<td>26</td>
</tr>
<tr>
<td>5.3.4</td>
<td>Parameters</td>
<td>27</td>
</tr>
<tr>
<td>5.3.5</td>
<td>Outputs</td>
<td>27</td>
</tr>
<tr>
<td>5.3.6</td>
<td>Generic Description</td>
<td>28</td>
</tr>
</tbody>
</table>
SimSWE Library Documentation

5.3.7 Simulink Implementation .............................................. 29

5.4 VerificationAndValidation .................................................. 30
  5.4.1 Purpose ................................................................. 30
  5.4.2 Component ............................................................. 30
  5.4.3 Inputs ................................................................. 30
  5.4.4 Parameters ............................................................. 31
  5.4.5 Outputs ................................................................. 31
  5.4.6 Generic Description .................................................. 31
  5.4.7 Simulink Implementation ............................................. 33

5.5 WorkBreakdown ............................................................... 34
  5.5.1 Purpose ................................................................. 34
  5.5.2 Component ............................................................. 34
  5.5.3 Inputs ................................................................. 34
  5.5.4 Parameters ............................................................. 34
  5.5.5 Outputs ................................................................. 35
  5.5.6 Generic Description .................................................. 35
  5.5.7 Simulink Implementation ............................................. 36

6 Generic Processes ............................................................. 37
  6.1 WorkTestRework ........................................................... 37
    6.1.1 Purpose ............................................................. 37
    6.1.2 Component .......................................................... 37
    6.1.3 Inputs ............................................................... 37
    6.1.4 Parameters .......................................................... 39
    6.1.5 Outputs ............................................................... 39
    6.1.6 Generic Description ................................................ 40
      6.1.6.1 Production ....................................................... 40
      6.1.6.2 Quality .......................................................... 41
      6.1.6.3 Control ......................................................... 42
    6.1.7 Simulink Implementation .......................................... 44

  6.2 Scrum ...................................................................... 47
    6.2.1 Purpose ............................................................. 47
    6.2.2 Component .......................................................... 47
    6.2.3 Inputs ............................................................... 47
    6.2.4 Parameters .......................................................... 48
    6.2.5 Outputs ............................................................... 50
    6.2.6 Generic Description ................................................ 50
      6.2.6.1 Top ............................................................... 50
      6.2.6.2 Subsystem FaultAccumulation .................................. 51
      6.2.6.3 Subsystem Sprint ............................................... 52
      6.2.6.4 Subsystem ResourceManagement ............................... 53
      6.2.6.5 Subsystem IterationControl ................................... 54
      6.2.6.6 Subsystem FaultManagement ................................... 55
      6.2.6.7 Subsystem DeliveryManagement ............................... 56

Version 1.1
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.6</td>
<td>DelayBasedResourceAllocation</td>
<td>79</td>
</tr>
<tr>
<td>7.6.1</td>
<td>Purpose</td>
<td>79</td>
</tr>
<tr>
<td>7.6.2</td>
<td>Component</td>
<td>79</td>
</tr>
<tr>
<td>7.6.3</td>
<td>Inputs</td>
<td>79</td>
</tr>
<tr>
<td>7.6.4</td>
<td>Parameters</td>
<td>80</td>
</tr>
<tr>
<td>7.6.5</td>
<td>Outputs</td>
<td>80</td>
</tr>
<tr>
<td>7.6.6</td>
<td>Generic Description</td>
<td>80</td>
</tr>
<tr>
<td>7.6.7</td>
<td>Simulink Implementation</td>
<td>81</td>
</tr>
<tr>
<td>7.7</td>
<td>BacklogBasedResourceDistribution</td>
<td>82</td>
</tr>
<tr>
<td>7.7.1</td>
<td>Purpose</td>
<td>82</td>
</tr>
<tr>
<td>7.7.2</td>
<td>Component</td>
<td>82</td>
</tr>
<tr>
<td>7.7.3</td>
<td>Inputs</td>
<td>82</td>
</tr>
<tr>
<td>7.7.4</td>
<td>Parameters</td>
<td>82</td>
</tr>
<tr>
<td>7.7.5</td>
<td>Outputs</td>
<td>83</td>
</tr>
<tr>
<td>7.7.6</td>
<td>Generic Description</td>
<td>83</td>
</tr>
<tr>
<td>7.7.7</td>
<td>Simulink Implementation</td>
<td>84</td>
</tr>
<tr>
<td>8</td>
<td>Utilities</td>
<td>85</td>
</tr>
<tr>
<td>8.1</td>
<td>LevelVariable</td>
<td>85</td>
</tr>
<tr>
<td>8.1.1</td>
<td>Purpose</td>
<td>85</td>
</tr>
<tr>
<td>8.1.2</td>
<td>Component</td>
<td>85</td>
</tr>
<tr>
<td>8.1.3</td>
<td>Inputs</td>
<td>85</td>
</tr>
<tr>
<td>8.1.4</td>
<td>Parameters</td>
<td>85</td>
</tr>
<tr>
<td>8.1.5</td>
<td>Outputs</td>
<td>86</td>
</tr>
<tr>
<td>8.1.6</td>
<td>Generic Description</td>
<td>86</td>
</tr>
<tr>
<td>8.1.7</td>
<td>Simulink Implementation</td>
<td>86</td>
</tr>
<tr>
<td>8.2</td>
<td>LevelVariableResetable</td>
<td>87</td>
</tr>
<tr>
<td>8.2.1</td>
<td>Purpose</td>
<td>87</td>
</tr>
<tr>
<td>8.2.2</td>
<td>Component</td>
<td>87</td>
</tr>
<tr>
<td>8.2.3</td>
<td>Inputs</td>
<td>87</td>
</tr>
<tr>
<td>8.2.4</td>
<td>Parameters</td>
<td>88</td>
</tr>
<tr>
<td>8.2.5</td>
<td>Outputs</td>
<td>88</td>
</tr>
<tr>
<td>8.2.6</td>
<td>Generic Description</td>
<td>88</td>
</tr>
<tr>
<td>8.2.7</td>
<td>Simulink Implementation</td>
<td>88</td>
</tr>
<tr>
<td>8.3</td>
<td>AsymmetricDelay</td>
<td>89</td>
</tr>
<tr>
<td>8.3.1</td>
<td>Purpose</td>
<td>89</td>
</tr>
<tr>
<td>8.3.2</td>
<td>Component</td>
<td>89</td>
</tr>
<tr>
<td>8.3.3</td>
<td>Inputs</td>
<td>89</td>
</tr>
<tr>
<td>8.3.4</td>
<td>Parameters</td>
<td>89</td>
</tr>
<tr>
<td>8.3.5</td>
<td>Outputs</td>
<td>90</td>
</tr>
<tr>
<td>8.3.6</td>
<td>Generic Description</td>
<td>90</td>
</tr>
<tr>
<td>8.3.7</td>
<td>Simulink Implementation</td>
<td>90</td>
</tr>
<tr>
<td>8.4</td>
<td>RateConversion</td>
<td>91</td>
</tr>
<tr>
<td>8.4.1</td>
<td>Purpose</td>
<td>91</td>
</tr>
</tbody>
</table>

Version 1.1
8.4.2 Component ...................................................... 91
8.4.3 Inputs ............................................................ 91
8.4.4 Parameters ...................................................... 91
8.4.5 Outputs .......................................................... 91
8.4.6 Generic Description ............................................ 92
8.4.7 Simulink Implementation ................................. 92

8.5 Size2Rate .......................................................... 93
8.5.1 Purpose ........................................................ 93
8.5.2 Component ...................................................... 93
8.5.3 Inputs ............................................................ 93
8.5.4 Parameters ...................................................... 93
8.5.5 Outputs .......................................................... 93
8.5.6 Generic Description ............................................ 94
8.5.7 Simulink Implementation ................................. 94

9 Examples ............................................................. 95
9.1 Effects of Different Prioritization Strategies ........................ 95
  9.1.1 Model name .................................................. 95
  9.1.2 SimSWE Components Used .................................. 95
  9.1.3 Purpose ...................................................... 95
  9.1.4 Simulink Implementation ................................. 95
9.2 Brooks Law ........................................................... 96
  9.2.1 Model name .................................................. 96
  9.2.2 SimSWE Components Used .................................. 96
  9.2.3 Purpose ...................................................... 96
  9.2.4 Simulink Implementation ................................. 96
9.3 V-Models with and without Error Propagation ..................... 97
  9.3.1 Model names .................................................. 97
  9.3.2 SimSWE Components Used .................................. 97
  9.3.3 Purpose ...................................................... 97
  9.3.4 Simulink Implementation ................................. 97
  9.3.5 Simulink Implementation ................................. 98
9.4 Scrum process ...................................................... 99
  9.4.1 Model names .................................................. 99
  9.4.2 SimSWE Components Used .................................. 99
  9.4.3 Purpose ...................................................... 99
  9.4.4 Simulink Implementation ................................. 99

II SimSWE (extended) .................................................. 100

10 Generators .......................................................... 101
  10.1 TaskGeneratorD ................................................. 101
    10.1.1 Purpose .................................................. 101

Version 1.1
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.1.2 Component</td>
<td>101</td>
</tr>
<tr>
<td>10.1.3 Inputs</td>
<td>101</td>
</tr>
<tr>
<td>10.1.4 Parameters</td>
<td>101</td>
</tr>
<tr>
<td>10.1.5 Outputs</td>
<td>102</td>
</tr>
<tr>
<td>10.1.6 Generic Description</td>
<td>102</td>
</tr>
<tr>
<td>10.1.7 Simulink Implementation</td>
<td>103</td>
</tr>
<tr>
<td>MultiSizeTaskGeneratorD</td>
<td>104</td>
</tr>
<tr>
<td>10.2.1 Purpose</td>
<td>104</td>
</tr>
<tr>
<td>10.2.2 Component</td>
<td>104</td>
</tr>
<tr>
<td>10.2.3 Inputs</td>
<td>104</td>
</tr>
<tr>
<td>10.2.4 Parameters</td>
<td>104</td>
</tr>
<tr>
<td>10.2.5 Outputs</td>
<td>106</td>
</tr>
<tr>
<td>10.2.6 Generic Description</td>
<td>106</td>
</tr>
<tr>
<td>10.2.7 Simulink Implementation</td>
<td>106</td>
</tr>
<tr>
<td>11 Estimators</td>
<td>107</td>
</tr>
<tr>
<td>12 Generic Activities</td>
<td>108</td>
</tr>
<tr>
<td>ProductionD</td>
<td>108</td>
</tr>
<tr>
<td>12.1.1 Purpose</td>
<td>108</td>
</tr>
<tr>
<td>12.1.2 Component</td>
<td>108</td>
</tr>
<tr>
<td>12.1.3 Inputs</td>
<td>108</td>
</tr>
<tr>
<td>12.1.4 Parameters</td>
<td>109</td>
</tr>
<tr>
<td>12.1.5 Outputs</td>
<td>109</td>
</tr>
<tr>
<td>12.1.6 Generic Description</td>
<td>109</td>
</tr>
<tr>
<td>12.1.7 Simulink Implementation</td>
<td>110</td>
</tr>
<tr>
<td>VerificationD</td>
<td>111</td>
</tr>
<tr>
<td>12.2.1 Purpose</td>
<td>111</td>
</tr>
<tr>
<td>12.2.2 Component</td>
<td>111</td>
</tr>
<tr>
<td>12.2.3 Inputs</td>
<td>111</td>
</tr>
<tr>
<td>12.2.4 Parameters</td>
<td>112</td>
</tr>
<tr>
<td>12.2.5 Outputs</td>
<td>112</td>
</tr>
<tr>
<td>12.2.6 Generic Description</td>
<td>112</td>
</tr>
<tr>
<td>12.2.7 Simulink Implementation</td>
<td>113</td>
</tr>
<tr>
<td>13 Generic Processes</td>
<td>114</td>
</tr>
<tr>
<td>WorkTestReworkD</td>
<td>114</td>
</tr>
<tr>
<td>13.1.1 Purpose</td>
<td>114</td>
</tr>
<tr>
<td>13.1.2 Component</td>
<td>114</td>
</tr>
<tr>
<td>13.1.3 Inputs</td>
<td>114</td>
</tr>
<tr>
<td>13.1.4 Parameters</td>
<td>115</td>
</tr>
<tr>
<td>13.1.5 Outputs</td>
<td>115</td>
</tr>
<tr>
<td>13.1.6 Generic Description</td>
<td>116</td>
</tr>
<tr>
<td>13.1.7 Simulink Implementation</td>
<td>116</td>
</tr>
</tbody>
</table>
1 Structure

The SimSWE library is partitioned into two sets

- **SimSWE (basic)**
  This set contains time based simulation blocks only.

- **SimSWE (extended)**
  This set contains also discrete event blocks.

Each set is structured into six subcollections: Generators, Estimators, Generic Activities, Generic Processes, Management Components, and Utilities, see Fig. 1.1.

Figure 1.1: SimSWE library in the Simulink Browser
2 Installation

2.1 Requirements for Simulink Libraries

The Simulink implementation of the library SimSWE (basic) uses the basic MATLAB® and Simulink® packages only. There are no additional toolboxes required.

The library SimSWE (extended) requires the toolboxes SimEvents® and Stateflow®.

2.2 Installation of Simulink Libraries

To use the Simulink implementation of the SimSWE libraries, the contents of the archive SimSWE_Simulink.zip need to be extracted into a directory.

This will create the subdirectories examples, simswe_basic, and simswe_extended. Using pathtool (type pathtool at the Matlab command prompt) the directories simswe_basic, and simswe_extended (as created in the step before) must be added to matlab search path by choosing the command Add with Subfolders....

After that, the Simulink Library Browser can be started and should display the two libraries.
Part I

SimSWE (basic)
3 Generators

3.1 RequirementsGenerator

3.1.1 Purpose

The component generates requirements for a project as a set of number-cost-value tuples. The requirements are generated at start and can be accessed via triggering the Next port.

The value of all requirements and the cost of unimplemented requirements are randomly updated during simulation according to volatility parameters. The output order of the requirements can be prioritized based on cost, value, or benefit (value per cost).

The component is based on [5].

3.1.2 Component

![Figure 3.1: RequirementsGenerator component](image)

3.1.3 Inputs

- **Start**
  
  Signal to trigger (rising edge) the (re)generation of requirements and to enable the cost-value update process.
• Next
  Trigger to access the next requirement (at the output ports).
  Unit: [ ]
  Values: \{0, 1\}

3.1.4 Parameters

• StepSize
  Sample time of computations (update, prioritization)
  Unit: [ time unit ]
  Range: \((0, \infty)\)

• VolatilityRate
  Time steps between updates of the requirements (cost, value). The cost is updated only for requirements, which have not yet been delivered at the output (i.e. implementation has not yet started).
  Unit: [ # ]
  Range: \([0, \infty)\)

• PrioritizationRate
  Time steps between reprioritization (reordering) of the requirements. Only requirements, which have not yet been accessed at the output (i.e. implementation has not yet started), are reordered.
  Unit: [ # ]
  Range: \([0, \infty)\)

• PrioritizationType
  1 = prioritization according to value (descending)
  2 = prioritization according to cost (ascending)
  3 = prioritization according to benefit (value per cost)

• Number
  Initial number of requirements at start.
  Unit: [ # ]
  Range: \([0, \infty)\)

• AvgNewReqs
  Average number of new requirements, which are generated at each update step. The actual number of new requirements is generated using a Poisson distribution.
  Unit: [ # ]
  Range: \([0, \infty)\)

• MinCost
Minimum cost of generated requirements, which are generated at start. The actual cost is generated using a uniform distribution.
Unit: [ size unit ]
Range: [0, $\infty$)

• **MaxCost**
  Maximum cost of generated requirements, which are generated at start. The actual cost is generated using a uniform distribution.
  Unit: [ size unit ]
  Range: [0, $\infty$)

• **CostSigma**
  Standard deviation of cost changes at each update step. The actual cost is generated using a normal distribution.
  Unit: [ size unit ]
  Range: [0, $\infty$)

• **MinValue**
  Minimum value of generated requirements, which are generated at start. The actual value is generated using a uniform distribution.
  Unit: [ value unit ]
  Range: [0, $\infty$)

• **MaxValue**
  Maximum value of generated requirements, which are generated at start. The actual value is generated using a uniform distribution.
  Unit: [ value unit ]
  Range: [0, $\infty$)

• **ValueSigma**
  Standard deviation of value changes at each update step. The actual value is generated using a normal distribution.
  Unit: [ value unit ]
  Range: [0, $\infty$)

### 3.1.5 Outputs

• **TotalReqs**
  Total number of generated requirements (at start and during updates).
  Unit: [ # ]
  Range: [0, $\infty$)

• **NewReqs**
  Number of new requirements at the current time step
  Unit: [ # ]
• **CurrentCost**  
  Cost of the current requirement.  
  Unit: [ size unit ]  
  Range: $[0, \infty)$

• **CurrentValue**  
  Value of the current requirement.  
  Unit: [ value unit ]  
  Range: $[0, \infty)$

• **Number**  
  Number of the current requirement according to the last prioritization.  
  Unit: [ # ]  
  Range: $[0, \infty)$

• **AccumulatedValue**  
  Accumulated value of all already accessed (i.e. implemented) requirements.  
  Unit: [ value unit ]  
  Range: $[0, \infty)$

### 3.1.6 Generic Description

Work in progress
3.1.7 Simulink Implementation

Figure 3.2: RequirementsGenerator implementation
3.2 FunctionPointGenerator

3.2.1 Purpose
The component generates function points for a project based on a set of characterising parameters.

3.2.2 Component
Under construction

3.2.3 Inputs
Under construction

3.2.4 Parameters
Under construction

3.2.5 Outputs
Under construction

3.2.6 Generic Description
Under construction
4 Estimators

4.1 COCOMOIIEarlyDesign

4.1.1 Purpose

This component is used to estimate a project’s cost and duration before the entire architecture is determined. It uses 7 Cost Driver and 5 Scale Factors. The component is based on COCOMO II [1].

4.1.2 Component

![Figure 4.1: COCOMOIIEarlyDesign component](image)

4.1.3 Inputs

- **KSLOC**
  Size of the project measured by thousand source line of codes
  Unit: [\#]
  Range: (0, ∞)

4.1.4 Parameters

Under construction

4.1.5 Outputs

- **Effort**
  Estimated effort expressed as person months.
• **Duration**
  Estimated project time expressed as months.
  Unit: [Month]
  Range: $[0, \infty)$

### 4.1.6 Generic Description

Under construction
4.2 COCOMOIIPostArchitecture

4.2.1 Purpose

This component is used to estimate a project’s cost and duration based on detailed architecture parameters. It uses 17 Cost Driver and 5 Scale Factors. The component is based on COCOMO II [1].

4.2.2 Component

![COCOMOIIPostArchitecture component](image)

Figure 4.2: COCOMOIIPostArchitecture component

4.2.3 Inputs

- **KSLOC**
  Size of the project measured by thousand source lines of code
  Unit: [#]
  Range: (0, ∞)

4.2.4 Parameters

Under construction

4.2.5 Outputs

- **Effort**
  Estimated effort expressed as person months.
  Unit: [Person Month]
  Range: [0, ∞)

- **Duration**
  Estimated project time expressed as months.
  Unit: [Month]
Range: $[0, \infty)$

4.2.6 Generic Description
Under construction

4.2.7 Simulink Implementation
Under construction
4.3 **SLOCEstimatorBasedOnFunctionPoints**

4.3.1 **Purpose**

This component estimates a project’s source lines of code (SLOC) based on unadjusted function points using programming language conversion factors. The component is based on COCOMO II [1].

4.3.2 **Component**

![SLOC Estimator Based on Function Points Component](image)

Figure 4.3: *SLOCEstimatorBasedOnFunctionPoints* component

4.3.3 **Inputs**

- *UnadjustedFunctionPoints*
  Project size measured by unadjusted function points.
  Unit: [#]
  Range: (0, ∞)

Version 1.1
4.3.4 Parameters

- *Language*

<table>
<thead>
<tr>
<th>Language</th>
<th>SLOC/FP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assembler</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
</tr>
<tr>
<td>Pascal</td>
<td>3</td>
</tr>
<tr>
<td>C++</td>
<td>4</td>
</tr>
<tr>
<td>Modula-2</td>
<td>5</td>
</tr>
<tr>
<td>Ada83</td>
<td>6</td>
</tr>
<tr>
<td>OO-Languages</td>
<td>7</td>
</tr>
<tr>
<td>4. Generation Languages</td>
<td>8</td>
</tr>
<tr>
<td>Visual Basic 5.0</td>
<td>9</td>
</tr>
<tr>
<td>HTML</td>
<td>10</td>
</tr>
<tr>
<td>Java</td>
<td>11</td>
</tr>
<tr>
<td>Visual C++</td>
<td>12</td>
</tr>
</tbody>
</table>

4.3.5 Outputs

- *KSLOC*
  Project size measured thousand (kilo) source lines of code.
  Unit: [#]
  Range: (0, ∞)

4.3.6 Generic Description

<table>
<thead>
<tr>
<th>Language</th>
<th>SLOC/FP</th>
</tr>
</thead>
<tbody>
<tr>
<td>4. Generation Languages</td>
<td>20</td>
</tr>
<tr>
<td>Assembler</td>
<td>320</td>
</tr>
<tr>
<td>Ada83</td>
<td>71</td>
</tr>
<tr>
<td>C</td>
<td>128</td>
</tr>
<tr>
<td>C++</td>
<td>55</td>
</tr>
<tr>
<td>Java</td>
<td>53</td>
</tr>
<tr>
<td>HTML</td>
<td>15</td>
</tr>
<tr>
<td>Modula-2</td>
<td>80</td>
</tr>
<tr>
<td>OO-Languages</td>
<td>29</td>
</tr>
<tr>
<td>Pascal</td>
<td>91</td>
</tr>
<tr>
<td>Visual Basic 5.0</td>
<td>29</td>
</tr>
<tr>
<td>Visual C++</td>
<td>34</td>
</tr>
</tbody>
</table>

Table 4.1: UFP to SLOC Conversion Ratios

The unadjusted function points have to be converted to source lines of code in the
implementation language (Ada, C, C++, Pascal, etc.). The block does this for both the Early Design and Post-Architecture models by using backfiring tables to convert Adjusted Function Points into equivalent SLOC. The current conversion ratios shown in Table 4.3.6 are from [2].
4.4 PerceivedQuality

4.4.1 Purpose

The component models the asymmetric dynamic of the perception of quality changes: In case of a decrease of quality, the perceived quality follows this change very fast. In case of an increase, the build-up in perceived quality is much slower. The model is based on [4].

4.4.2 Component

![PerceivedQuality component](image)

Figure 4.4: PerceivedQuality component

4.4.3 Inputs

- **IndicatedQuality**
  This signal is the actual indicated software quality.
  Unit: \([\text{qualityunit}]\)
  Range: \((-\infty, \infty)\)

4.4.4 Parameters

- **InitialPerceivedQuality**
  Initial perceived quality at the beginning of the simulation.
  Unit: \([\text{qualityunit}]\)
  Range: \((-\infty, \infty)\)

- **DelayGrowth**
  Is the delay time constant for the improvement process.
  Unit: \([\text{timeunit}]\)
  Range: \((0, \infty)\)

- **SpeedUpFactor**
  The time constant for the decrease is the time constant of the increase divided by this factor, i.e., SpeedUpFactor controls the difference in the dynamics of the
increase and the decrease.
Unit: [ ]
Range: (0, ∞)

4.4.5 Outputs

- \( \text{PerceivedQuality} \)
  This signal is the actual perceived software quality.
  Unit: [\text{qualityunit}]
  Range: (−∞, ∞)

4.4.6 Generic Description

\[
\frac{d}{dt} \text{PerceivedQuality} = \begin{cases} 
\frac{\text{IndicatedQuality} - \text{PerceivedQuality}}{\text{DelayGrowth}} & \text{if } \text{IndicatedQuality} - \text{PerceivedQuality} > 0 \\
(\text{IndicatedQuality} - \text{PerceivedQuality}) \cdot \frac{\text{SpeedUpFactor}}{\text{DelayGrowth}} & \text{if } \text{IndicatedQuality} - \text{PerceivedQuality} \leq 0
\end{cases}
\] (4.1)

\( \text{PerceivedQuality}(0) = \text{InitialPerceivedQuality} \)

4.4.7 Simulink Implementation

![Simulink Diagram]

Figure 4.5: \( \text{PerceivedQuality} \) implementation
5 Generic Activities

5.1 Work

5.1.1 Purpose

The Work component models the work on a given artifact or task (e.g. development) based on task size and productivity. The model is based on concepts of [3] and [4].

5.1.2 Component

![Work component diagram](image)

Figure 5.1: Work component

5.1.3 Inputs

- **Productivity**
  Productivity available for the task.
  Unit: [ size unit / time unit ]
  Range: (0, ∞)

- **TaskSizeRate**
  Rate of the designated size of the task or artifact. The size is the integral of the rate between two resets. TaskSizeRate enables modeling a build up of a task size over a time span.
  Unit: [ size unit / time unit ]
  Range: [0, ∞)

- **Reset**
  Reset of task used to start a new task. The reset trigger is the rising edge of this
5.1.4 Parameters

- **AverageFaultInjectionPerSizeUnit**
  Number of faults injected during the initial development phase (note, currently there is no random part in the fault injection).
  Unit: [ # / size unit ]
  Range: [0, ∞)

5.1.5 Outputs

- **DevelopmentRate**
  Actual development rate.
  Unit: [ size unit / time unit ]
  Range: [0, ∞)

- **ProducedSize**
  Part of the task, which is already finished. The task size is the sum of `ProducedSize` and `ToProduceSize`.
  Unit: [ size unit ]
  Range: [0, ∞)

- **ToProduceSize**
  Part of the task, which is not yet finished. The task size is the sum of `ProducedSize` and `ToProduceSize`.
  Unit: [ size unit ]
  Range: [0, ∞)

- **FaultRate**
  Rate of the faults produced.
  Unit: [ # / time unit ]
  Range: [0, ∞)

- **Faults**
  Number of faults contained in the work product.
  Unit: [ # ]
  Range: [0, ∞)

5.1.6 Generic Description

\[
DevelopmentRate = \begin{cases} 
  \text{Productivity} & \text{if } ToProduceSize > 0 \\
  \min(\text{Productivity}, \text{TaskSizeRate}) & \text{otherwise}
\end{cases}
\]
\[ \frac{d}{dt} \text{TaskSize} = \text{TaskSizeRate} \] 
(5.1)

Reset: \( \text{TaskSize} = 0 \) at \( t = \text{ResetEvent} \)

\( \text{ResetEvent} \): Rising edge of \( \text{Reset} \)

\[ \frac{d}{dt} \text{ProducedSize} = \text{DevelopmentRate} \] 
(5.2)

Reset: \( \text{ProducedSize} = 0 \) at \( t = \text{ResetEvent} \)

\( \text{ResetEvent} \): Rising edge of \( \text{Reset} \)

\[ \text{ToProduceSize} = \text{TaskSize} - \text{ProducedSize} \] 
(5.3)

\[ \text{FaultRate} = \text{AverageFaultInjectionPerSizeUnit} \cdot \text{DevelopmentRate} \] 
(5.4)

\[ \frac{d}{dt} \text{Faults} = \text{FaultRate} \] 
(5.5)

Reset: \( \text{Faults} = 0 \) at \( t = \text{ResetEvent} \)

\( \text{ResetEvent} \): Rising edge of \( \text{Reset} \)

5.1.7 Simulink Implementation

![Simulink Diagram](image)

Figure 5.2: Work implementation
5.2 WorkWithCommunicationOverhead

5.2.1 Purpose

The WorkWithCommunicationOverhead component models the work on a given task or artifact (e.g. development) considering communication overhead for adjusting the effective productivity. The model is based on concepts of [3] and [4].

5.2.2 Component

![Component Diagram]

Figure 5.3: WorkWithCommunicationOverhead component

5.2.3 Inputs

- **RawProductivity**
  Productivity without communication overhead.
  Unit: [ size unit / time unit ]
  Range: [0, ∞)

- **NumberOfPersons**
  Total number of persons available.
  Unit: [ # ]
  Range: [0, ∞)

- **TaskSizeRate**
  Rate of the designated size of the task or artifact. The size is the integral of the rate between two resets. TaskSizeRate enables modeling a build up of a task size over a time span.
  Unit: [ size unit / time unit ]
  Range: [0, ∞)

- **Reset**
  Reset of task used to start a new task. The reset trigger is the rising edge of this
signal.
Unit: [ ]
Values: \{0, 1\}

5.2.4 Parameters

- **AverageFaultInjectionPerSizeUnit**
  Number of faults injected during the initial development phase (note, currently there is no random part in the fault injection).
  Unit: [ \# / size unit ]
  Range: \([0, \infty)\)

- **RelativeCommunicationOverhead**
  Relative communication overhead.
  Unit: [1 / \#^2]
  Range: \([0, \infty)\)

5.2.5 Outputs

- **DevelopmentRate**
  Actual development rate.
  Unit: [ size unit / time unit ]
  Range: \((0, \infty)\)

- **AdjustedProductivity**
  Actual productivity taking into account the communication overhead.
  Unit: [sizeunit/timeunit]
  Range: \([0, \infty)\)

- **ProducedSize**
  Part of the task, which is already finished. The task size is the sum of ProducedSize and ToProduceSize.
  Unit: [ size unit ]
  Range: \([0, \infty)\)

- **ToProduceSize**
  Part of the task, which is not yet finished. The task size is the sum of ProducedSize and ToProduceSize.
  Unit: [ size unit ]
  Range: \([0, \infty)\)

- **FaultRate**
  Rate of the faults produced.
  Unit: [ \# / time unit ]
  Range: \([0, \infty)\)

- **Faults**
  Number of faults contained in the work product.
5.2.6 Generic Description

The component WorkWithCommunicationOverhead consists of a component CommunicationOverhead, see Sec. 7.2 and a component Work, see Sec. 5.1.

WorkWithCommunicationOverhead outputs:

\[
\begin{align*}
\text{DevelopmentRate} &= Work.DevelopmentRate \quad (5.7) \\
\text{AdjustedProductivity} &= CommunicationOverhead.AdjustedProductivity \quad (5.8) \\
\text{ProducedSize} &= Work.ProducedSize \quad (5.9) \\
\text{ToProduceSize} &= Work.ToProduceSize \quad (5.10) \\
\text{FaultRate} &= Work.FaultRate \quad (5.11) \\
\text{Faults} &= Work.Faults \quad (5.12)
\end{align*}
\]

CommunicationOverhead inputs:

\[
\begin{align*}
\text{CommunicationOverhead.RawProductivity} &= RawProductivity \quad (5.13) \\
\text{CommunicationOverhead.NumberOfPersons} &= NumberOfPersons \quad (5.14)
\end{align*}
\]

CommunicationOverhead parameters:

\[
\begin{align*}
\text{CommunicationOverhead.RelativeCommunicationOverhead} &= RelativeCommunicationOverhead \quad (5.15)
\end{align*}
\]

Work inputs:

\[
\begin{align*}
\text{Work.Productivity} &= AdjustedProductivity \quad (5.16) \\
\text{Work.TaskSizeRate} &= TaskSizeRate \quad (5.17) \\
\text{Work.Reset} &= Reset \quad (5.18)
\end{align*}
\]

Work parameters:

\[
\begin{align*}
\text{Work.AverageFaultInjectionPerSizeUnit} &= AverageFaultInjectionPerSizeUnit \quad (5.19)
\end{align*}
\]
5.2.7 Simulink Implementation

Figure 5.4: WorkWithCommunicationOverhead implementation
5.3 WorkWithExperience

5.3.1 Purpose

The WorkWithExperience component models the work on a given task or artifact (e.g. development) considering experience for adjusting the effective productivity. The model is based on concepts of [3] and [4].

5.3.2 Component

![WorkWithExperience component](image)

Figure 5.5: WorkWithExperience component

5.3.3 Inputs

- **NumberOfPersons**
  The number of persons influence the productivity and the experiencing rate.
  Unit: [ # ]
  Range: [0, ∞)

- **TaskSizeRate**
  Rate of the designated size of the task or artifact. The size is the integral of the rate between two resets. TaskSizeRate enables modeling a build up of a task size over a time span.
  Unit: [ size unit / time unit ]
  Range: [0, ∞)

- **Reset**
  An rising edge on this input port resets the remaining work and the experience level from the actual project to zero. The experience from similar projects in the past does not change.
  Unit: [ ]
5.3.4 Parameters

- **AverageFaultInjectionPerSizeUnit**
  Number of faults injected during the initial development phase (note, currently there is no random part in the fault injection).
  Unit: [ # / size unit ]
  Range: $[0, \infty)$

- **LearningRate**
  The learning rate influence the experiencing speed. Higher learning rates slows down and lower learning rates speeds up the experiencing rate.
  Unit: [ % ]
  Range: $[0, 100]$%

- **FirstUnitCost**
  Required effort for the first unit. In general the required effort for the next unit drops down.
  Unit: [(time unit · #) / size unit]
  Range: $(0, \infty)$

- **MinUnitCost**
  Smallest achievable value for the required effort per unit. This value has to be smaller then the required effort for the first unit.
  Unit: [(time unit · #) / size unit]
  Range: $(0, \infty)$

- **TransferredExperience**
  Transferred experience from similar projects in the past. This parameter is available only, if the option S-Curve is active.
  Unit: [ size unit ]
  Range: $[0, \infty)$

5.3.5 Outputs

- **DevelopmentRate**
  Actual development rate.
  Unit: [ size unit / time unit ]
  Range: $(0, \infty)$

- **AdjustedProductivity**
  Productivity taking into account the experience build-up.
  Unit: [ size unit / time unit ]
  Range: $[0, \infty)$

- **ProducedSize**
  Part of the task, which is already finished. The task size is the sum of ProducedSize
and $ToProduceSize$.
Unit: [ size unit ]
Range: $[0, \infty)$

- $ToProduceSize$
Part of the task, which is not yet finished. The task size is the sum of $ProducedSize$ and $ToProduceSize$.
Unit: [ size unit ]
Range: $[0, \infty)$

- $FaultRate$
Rate of the faults produced.
Unit: [ # / time unit ]
Range: $[0, \infty)$

- $Faults$
Number of faults contained in the work product.
Unit: [ # ]
Range: $[0, \infty)$

- $Experience$
Is the actual experience level. This level depends on done work and transferred experience from similar done projects in the past.
Unit: [ size unit ]
Range: $[0, \infty)$

### 5.3.6 Generic Description

The component $WorkWithExperience$ consists of a component $ExperienceAcquisition$, see Sec. 7.4 and a component $Work$, see Sec. 5.1.

$WorkWithExperience$ outputs:

\[
DevelopmentRate = Work.DevelopmentRate \tag{5.20}
\]

\[
Productivity = ExperienceAcquisition.Productivity \tag{5.21}
\]

\[
ProducedSize = Work.ProducedSize \tag{5.22}
\]

\[
ToProduceSize = Work.ToProduceSize \tag{5.23}
\]

\[
FaultRate = Work.FaultRate \tag{5.24}
\]

\[
Faults = Work.Faults \tag{5.25}
\]

\[
Experience = ExperienceAcquisition.Experience \tag{5.26}
\]

$ExperienceAcquisition$ inputs:

\[
ExperienceAcquisition.ExperiencingRate = DevelopmentRate \tag{5.27}
\]

\[
ExperienceAcquisition.NumberOfPersons = NumberOfPersons \tag{5.28}
\]

\[
ExperienceAcquisition.EnableExperiencing = 1 \tag{5.29}
\]

\[
ExperienceAcquisition.Reset = Reset \tag{5.30}
\]
**ExperienceAcquisition** parameters:

\[
\text{ExperienceAcquisition.ExperienceCurve} = S - \text{Curve} \quad (5.31)
\]
\[
\text{ExperienceAcquisition.LearningRate} = \text{LearningRate} \quad (5.32)
\]
\[
\text{ExperienceAcquisition.FirstUnitCost} = \text{FirstUnitCost} \quad (5.33)
\]
\[
\text{ExperienceAcquisition.MinUnitCost} = \text{MinUnitCost} \quad (5.34)
\]
\[
\text{ExperienceAcquisition.TransferredExperience} = \text{TransferredExperience} \quad (5.35)
\]

**Work** inputs:

\[
\text{Work.Productivity} = \text{Productivity} \quad (5.36)
\]
\[
\text{Work.TaskSizeRate} = \text{TaskSizeRate} \quad (5.37)
\]
\[
\text{Work.Reset} = \text{Reset} \quad (5.38)
\]

**Work** parameters:

\[
\text{Work.AverageFaultInjectionPerSizeUnit} = \frac{\text{AverageFaultInjectionPerSizeUnit}}{} \quad (5.39)
\]

### 5.3.7 Simulink Implementation

![Simulink Diagram](image)

Figure 5.6: *WorkWithExperience* implementation
5.4 VerificationAndValidation

5.4.1 Purpose

Model of a verification and validation (V&V) activity. The model is based on concepts of [3].

5.4.2 Component

![VerificationAndValidation component](image)

Figure 5.7: VerificationAndValidation component

5.4.3 Inputs

- **VerificationProductivity**
  
  Productivity of the V&V activity.
  
  Unit: [ size unit / time unit ]
  
  Range: (0, \( \infty \))

- **SizeRate**
  
  Rate of the size of the V&V task. The size is the integral of the rate between two resets. TaskSizeRate enables modeling a build up of a task size over a time span.
  
  Unit: [ size unit / time unit ]
  
  Range: \([0, \infty)\)

- **UndetectedFaultsRate**
  
  Rate of the faults with the task or artifact received. The number of undetected faults is the integral of the rate between two resets.
  
  Unit: [ # / time unit ]
  
  Range: \([0, \infty)\)

- **Reset**
  
  Reset of task used to start a new task. The reset trigger is the rising edge of this signal.
  
  Unit: [ ]
  
  Values: \(\{0, 1\}\)
5.4.4 Parameters

- **AverageVerEffectiveness**
  Parameter controlling the effectiveness of the V&V activity.
  Unit [ % ]
  Range: (0, 100]

5.4.5 Outputs

- **Verification rate**
  Rate of the V&V process.
  Unit [ size unit / time unit]
  Range: [0, ∞)

- **VerifiedSize**
  Size already processed.
  Unit [ size unit ]
  Range: [0, ∞)

- **ToVerifySize**
  Size not yet processed.
  Unit [ size unit ]
  Range: [0, ∞)

- **FaultsDetectedRate**
  Rate of the fault detection.
  Unit [ # / time unit ]
  Range: [0, ∞)

- **FaultsDetectedRate**
  Number of detected faults.
  Unit [ # ]
  Range: [0, ∞)

5.4.6 Generic Description

\[ \frac{d}{dt} \text{Size} = \text{SizeRate} \] (5.40)
Reset : \( \text{Size} = 0 \) at \( t = \text{ResetEvent} \)
\( \text{ResetEvent} \) : Rising edge of \( \text{Reset} \)

\[ \frac{d}{dt} \text{FaultsUndetected} = \text{UndetectedFaultsRate} - \text{FaultsDetectedRate} \] (5.41)
Reset : \( \text{FaultsUndetected} = 0 \) at \( t = \text{ResetEvent} \)
\( \text{ResetEvent} \) : Rising edge of \( \text{Reset} \)
\[
\frac{d}{dt} \text{FaultsDetected} = \text{FaultsDetectedRate} \tag{5.42}
\]
Reset: \( \text{FaultsDetected} = 0 \) at \( t = \text{ResetEvent} \)
\( \text{ResetEvent} \): Rising edge of \( \text{Reset} \)

\[
\frac{d}{dt} \text{VerifiedSize} = \text{VerificationRate} \tag{5.43}
\]
Reset: \( \text{VerifiedSize} = 0 \) at \( t = \text{ResetEvent} \)
\( \text{ResetEvent} \): Rising edge of \( \text{Reset} \)

\[
\text{VerificationRate} =
\begin{cases} 
\text{VerificationProductivity} & \text{if } \text{ToVerifySize} > 0 \\
\min(\text{VerificationProductivity}, \text{SizeRate}) & \text{otherwise}
\end{cases} \tag{5.44}
\]

\[
\text{ToVerifySize} = \text{Size} - \text{VerifiedSize} \tag{5.45}
\]

\[
\text{FaultsDetectedRate} =
\begin{cases} 
f d & \text{if } (\text{Size} > 0) \land (\text{FaultsUndetected} > 0) \\
0 & \text{otherwise}
\end{cases} \tag{5.46}
\]

\[
f d = \frac{\text{AverageVerEffectiveness}}{100} \cdot \text{VerificationActivity} \cdot
derivation{\text{VerificationActivity} -}
\frac{(\text{FaultsUndetected} + \text{FaultsDetected})/\text{Size}}{}; \tag{5.47}
\]
5.4.7 Simulink Implementation

Figure 5.8: VerificationAndValidation implementation
5.5 WorkBreakdown

5.5.1 Purpose

Break down of a task into pieces (smaller tasks).

5.5.2 Component

![Diagram of WorkBreakdown component]

Figure 5.9: WorkBreakdown component

5.5.3 Inputs

- **SizeRateWhole**
  Input rate of the task, which is to be broken down. The size of the task is the integral of the rate between two resets.
  Unit: [ size unit / time unit ]
  Range: (−∞, ∞)

- **Reset**
  Reset of the component (reset at rising edge)
  Unit: [ ]
  Range: {0, 1}

- **Next**
  Trigger of the release of the next part (trigger at rising edge)
  Unit: [ ]
  Range: {0, 1}

5.5.4 Parameters

- **MinSize**
  Minimum size of a part
  Unit: [ size unit ]
  Range: [0, ∞)
• **MaxSize**
  Maximum size of a part
  Unit: [ size unit ]
  Range: [0, ∞)

• **Seed**
  Seed (of random generator)
  Unit: [ ]
  Range: (−∞, ∞)

• **ConversionRate**
  Magnitude of the delivery rate.
  Unit: [ size unit / time unit ]
  Range: (0, ∞)

### 5.5.5 Outputs

• **SizeRatePart**
  Size rate of the delivered parts
  Unit: [ size unit / time unit ]
  Range: [0, ∞)

• **SizePart**
  Size of the delivered parts
  Unit: [ size unit ]
  Range: [0, ∞)

• **RemainingSize**
  Fraction of the task not yet delivered as parts
  Unit: [ size unit ]
  Range: [0, ∞)

### 5.5.6 Generic Description

The component uses a component *LevelVariableResetable*, see Sec. 8.2 and a component *Size2Rate*, see Sec. 8.5.

*LevelVariableResetable* inputs:

\[
\text{LevelVariableResetable.In} = \text{SizeRateWhole} \quad (5.48)
\]
\[
\text{LevelVariableResetable.Reset} = \text{Reset} \quad (5.49)
\]
\[
\text{LevelVariableResetable.Initial} = 0 \quad (5.50)
\]
\[
\text{LevelVariableResetable.Out} = \text{SizeRatePart} \quad (5.51)
\]

*Size2Rate* inputs:

\[
\text{Size2Rate.Size} = \text{SizePart} \quad (5.52)
\]
\[
\text{Size2Rate.Reset} = \text{Next} \quad (5.53)
\]
Size2Rate parameters:

\[ \text{Size2Rate.ConversionRate} = \text{ConversionRate} \]  

(5.54)

Top-level logic:

\[ \text{SizeRatePart} = \text{Size2Rate.SizeRatePart} \]  

(5.55)

\[ \text{RemainingSize} = \text{LevelVariableResetable.Level} \]  

(5.56)

\[ \text{SizePart} = \min(\text{RandomNumber, RemainingSize}) \text{ at } t = \text{TriggerEvent} \]  

(5.57)

\text{TriggerEvent} : \text{Rising edge of Next}

\text{RandomNumber} is a random variable uniformly distributed in the interval \([\text{MinSize}, \text{MaxSize}]\) with seed \text{Seed}.

5.5.7 Simulink Implementation

![Simulink Diagram]

Figure 5.10: WorkBreakdown implementation
6 Generic Processes

6.1 WorkTestRework

6.1.1 Purpose

The WorkTestRework component models the production of an artifact followed by an associated verification and validation (V&V) step. If the artifact does not yet meet the specified quality criterion, it is scheduled for rework. Rework and evaluation are cyclically traversed until the quality criterion is fulfilled. In this case, the artifact is released.

The model is based on [3].

6.1.2 Component

![WorkTestRework component](image)

Figure 6.1: WorkTestRework component

6.1.3 Inputs

- **SizeReceived**
  Rate of the designated size of the artifact. This rate is determined by a previous process step. The size of the artifact is the integral of the rate between two releases. The SizeReceived rate allows modeling a task build-up over a time span.
  Unit: [ size unit / time unit ]
  Range: \([0, \infty)\)
• **UndetectedFaultsReceived**
  Rate of the undetected errors in the artifact caused by undetected errors in the precursor artifact (e.g. undetected errors in the design will cause undetected errors in the implementation). The number of undetected errors in the artifact caused by this source is the integral of the rate between two releases.
  Unit: [ # / time unit ]
  Range: $[0, \infty)$

• **DetectedFaultsReceived**
  Rate of the known errors in the artifact detected or caused by other process steps. The number of detected errors in the artifact caused by this source is the integral of the rate between two releases.
  Unit: [ # / time unit ]
  Range: $[0, \infty)$

• **DevelopmentProductivity**
  Net productivity of the initial development (work). Changes of the productivity (e.g. caused by staff allocation, training, motivation etc.) had to be modeled externally.
  Unit: [ size unit / time unit ]
  Range: $(0, \infty)$

• **VerificationProductivity**
  Net productivity of the verification. Changes of the productivity (e.g. caused by staff allocation, training, motivation etc.) have to be modeled externally.
  Unit: [ size unit / time unit ]
  Range: $(0, \infty)$

• **DeliveryProductivity**
  Net productivity of the delivery. Changes of the productivity (e.g. caused by staff allocation, training, motivation etc.) have to be modeled externally.
  Unit: [ size unit / time unit ]
  Range: $(0, \infty)$

• **RemovalProductivity**
  Net productivity of the error removal (rework). Changes of the productivity (e.g. caused by staff allocation, training, motivation etc.) have to be modeled externally.
  Unit: [ # / time unit ]
  Range: $(0, \infty)$

• **Reset**
  Reset of the component (reset at rising edge)
  Unit: [ ]
  Range: $\{0, 1\}$
6.1.4 Parameters

- **AverageVerEffectiveness**
  Parameter controlling the effectiveness of the V&V step.
  Unit: [ % ]
  Range: (0, 100]

- **QualityLimit**
  Release threshold with respect to the quotient of number of faults pending divided by the size of the artifact.
  Unit: [ # / size unit ]
  Range: (0, ∞)

- **AverageFaultInjectionPerSizeUnit**
  Number of faults injected during the initial development phase.
  Unit: [ # / size unit ]
  Range: [0, ∞)

- **AverageFaultInjectionPerFaultRemoval**
  Number of faults injected during the rework phase.
  Unit: [ # / # ]
  Range: [0, ∞)

6.1.5 Outputs

- **SizeDelivered**
  Rate of the size of the artifact delivered as a release. The rate is determined by the artifact size and the delivery productivity. Note: An artifact is delivered only when the test-rework-cycle has achieved the desired quality limit.
  Unit: [ size unit / time unit ]
  Range: [0, ∞)

- **UndetectedFaultsDelivered**
  Rate of the undetected errors in the artifact delivered.
  Unit: [ # / time unit ]
  Range: [0, ∞)

- **DetectedFaultsDelivered**
  Rate of the known errors in the artifact delivered.
  Unit: [ # / time unit ]
  Range: [0, ∞)

- **ToDeliverSize**
  Designated size of the artifact.
  Unit: [ Size unit ]
  Range: [0, ∞)

- **FaultsPending**
  Current number of faults pending.
Unit: [#]  
Range: $[0, \infty)$

- **StateIndex**
  Variable indicating the internal process state.
  0: Idle  
  1: Work  
  2: Rework  
  3: Verification  
  4: Delivery

### 6.1.6 Generic Description

#### 6.1.6.1 Production

\[
\text{DevelopmentActivityLimited} = \begin{cases} 
\text{DevelopmentActivity} & \text{if } \text{ToProduceSize} > 0 \\
\min(\text{SizeReceived}, \text{DevelopmentActivity}) & \text{otherwise} 
\end{cases} 
\]  
\tag{6.1}

\[
\frac{d}{dt}\text{ToProduceSize} = \text{SizeReceived} - \text{DevelopmentActivityLimited} 
\]  
\tag{6.2}

Reset: Level = 0 at $t = \text{ResetEvent}$  
\text{ResetEvent} : \text{Rising edge of Reset}

\[
\frac{d}{dt}\text{ToDeliverSize} = \text{DevelopmentActivityLimited} - \text{DeliveryActivity} 
\]  
\tag{6.3}

Reset: Level = 0 at $t = \text{ResetEvent}$  
\text{ResetEvent} : \text{Rising edge of Reset}

\[
\frac{d}{dt}\text{ToVerifySize} = \text{DevelopmentActivityLimited} - \text{VerificationActivity} 
\]  
\tag{6.4}

Reset: $\text{ToVerifySize} = \text{ToDeliverSize}$ at $t = \text{ResetEvent}$  
\text{ResetEvent} : \text{Rising edge of (Removal } \lor \text{ Idle } \lor \text{ Work)
6.1.6.2 Quality

\[ \frac{d}{dt} \text{FaultsUndetected} = \]
\[ \begin{align*} 
& \text{UndetectedFaultsReceived} \\
& + \text{AverageFaultInjectionPerSizeUnit} \cdot \text{DevelopmentActivity} \\
& + \text{AverageFaultInjectionPerFaultRemoval} \cdot \text{RemovalActivity} \\
& - \text{FaultDetection} - \text{UndetectedFaultsDelivered} 
\end{align*} \]  

\text{(6.5)}

Reset: Level = 0 at \( t = \text{ResetEvent} \)
ResetEvent: Rising edge of Reset

\[ \frac{d}{dt} \text{FaultsDetected} = \text{FaultDetection} \]  

\text{(6.6)}

Reset: \( \text{FaultsDetected} = 0 \) at \( t = \text{ResetEvent} \)
ResetEvent: Rising edge of Verification

\[ \text{FaultDetection} = \begin{cases} 
\text{fd} & \text{if } (\text{ToDeliverSize} > 0) \land (\text{FaultsUndetected} > 0) \\
0 & \text{otherwise} 
\end{cases} \]  

\text{(6.7)}

\[ \text{fd} = \frac{\text{AverageVerEffectiveness}}{100} \cdot \text{VerificationActivity} \cdot \\
\frac{(\text{FaultsUndetected} + \text{FaultsDetected})/\text{ToDeliverSize}}{}; \]  

\text{(6.8)}

\[ \frac{d}{dt} \text{FaultsPending} = \text{DetectedFaultsReceived} + \text{FaultDetection} \\
- \text{RemovalActivity} - \text{DetectedFaultsDelivered} \]  

\text{(6.9)}

Reset: Level = 0 at \( t = \text{ResetEvent} \)
ResetEvent: Rising edge of Reset

\[ \text{UndetectedFaultsDelivered} = \frac{\text{UndetectedFaultsToDeliver} \cdot \text{DeliveryActivity}}{\text{ToDeliverSize}} \]  

\text{(6.10)}

\[ \frac{d}{dt} \text{UndetectedFaultsToDeliver} = -\text{UndetectedFaultsDelivered} \]  

\text{(6.11)}

Reset: \( \text{UndetectedFaultsToDeliver} = \text{FaultsUndetected} \) at \( t = \text{ResetEvent} \)
ResetEvent: Rising edge of DeliveryActivity \( \neq 0 \)
\[ \text{DetectedFaultsDelivered} = \frac{\text{DetectedFaultsToDeliver} \cdot \text{DeliveryActivity}}{\text{ToDeliverSize}} \] (6.12)

\[ \frac{d}{dt} \text{DetectedFaultsToDeliver} = -\text{DetectedFaultsDelivered} \] (6.13)

Reset: \( \text{DetectedFaultsToDeliver} = \text{FaultsPending at } t = \text{ResetEvent} \)
ResetEvent: Rising edge of \( \text{DeliveryActivity} \neq 0 \)

### 6.1.6.3 Control

State machine see Fig. 6.2

\[ \text{ArtefactQuality} = \begin{cases} \text{FaultsPending} & \text{if } \text{ToDeliverSize} - \text{ToVerifySize} > 0 \\ \frac{\text{ToDeliverSize} - \text{ToVerifySize}}{\text{ToDeliverSize}} & \text{otherwise} \end{cases} \] (6.14)

\[ \text{DevelopmentActivity} = \begin{cases} \text{DevelopmentProductivity} & \text{if } \text{state} = \text{Work} \\ 0 & \text{otherwise} \end{cases} \] (6.15)

\[ \text{RemovalActivity} = \begin{cases} \text{RemovalProductivity} & \text{if } \text{state} = \text{Rework} \\ 0 & \text{otherwise} \end{cases} \] (6.16)

\[ \text{VerificationActivity} = \begin{cases} \text{VerificationProductivity} & \text{if } \text{state} = \text{Verification} \\ 0 & \text{otherwise} \end{cases} \] (6.17)

\[ \text{DeliveryActivity} = \begin{cases} \text{DeliveryProductivity} & \text{if } \text{state} = \text{Delivery} \\ 0 & \text{otherwise} \end{cases} \] (6.18)

\[ \text{SizeDelivered} = \text{DeliveryActivity} \] (6.19)

\[ \text{StateIndex} = \begin{cases} 0 & \text{if } \text{state} = \text{Idle} \\ 1 & \text{if } \text{state} = \text{Work} \\ 2 & \text{if } \text{state} = \text{Rework} \\ 3 & \text{if } \text{state} = \text{Verification} \\ 4 & \text{if } \text{state} = \text{Delivery} \end{cases} \] (6.20)
Figure 6.2: State machine controlling the WorkTestRework component
6.1.7 Simulink Implementation

Figure 6.3: WorkTestRework implementation

Figure 6.4: Production subsystem
Figure 6.5: Quality subsystem

Figure 6.6: Delivery subsystem
Figure 6.7: Control subsystem
6.2 Scrum

6.2.1 Purpose

Generic model of an iterative process (SCRUM). Incoming tasks are split into assignments with a fixed time length (sprints). At the start of each assignment (sprint), the overall amount is broken down into tasks. Each task is performed as a Work-Test-Rework cycle. Unfinished tasks are rescheduled for the next assignment.

6.2.2 Component

![Scrum component](image)

6.2.3 Inputs

- **SizeReceived**
  
  Rate of the designated size of the artifact. This rate is determined by a previous process step. The size of the artifact is the integral of the rate between two releases. The SizeReceived rate allows modeling a task build-up over a time span.
  
  Unit: [ size unit / time unit ]
  
  Range: [0, ∞)

- **DevelopmentProductivity**
  
  Net productivity of the initial development (work). Changes of the productivity (e.g. caused by staff allocation, training, motivation etc.) had to be modeled externally.
  
  Unit: [ size unit / time unit ]
  
  Range: (0, ∞)
• **VerificationProductivity**
  Net productivity of the verification. Changes of the productivity (e.g. caused by staff allocation, training, motivation etc.) have to be modeled externally.
  Unit: [ size unit / time unit ]
  Range: (0, ∞)

• **DeliveryProductivity**
  Net productivity of the delivery. Changes of the productivity (e.g. caused by staff allocation, training, motivation etc.) have to be modeled externally.
  Unit: [ size unit / time unit ]
  Range: (0, ∞)

• **RemovalProductivity**
  Net productivity of the error removal (rework). Changes of the productivity (e.g. caused by staff allocation, training, motivation etc.) have to be modeled externally.
  Unit: [ # / time unit ]
  Range: (0, ∞)

• **Reset**
  Reset of the component (reset at rising edge)
  Unit: [ ]
  Range: {0, 1}

• **EnableSprint**
  Enabling of the sprint iterations
  Unit: [ ]
  Range: {0, 1}

### 6.2.4 Parameters

• **SprintLength**
  Length of a sprint.
  Unit[ time unit ]
  Range: (0, ∞)

• **PlanningLength**
  Length of the planning at the beginning of a sprint.
  Unit[ time unit ]
  Range: (0, \(SprintLength\))

• **MinSizeSprint**
  Minimum size of a sprint backlog.
  Unit[ size unit ]
  Range: (0, ∞)

• **MaxSizeSprint**
  Maximum size of a sprint backlog.
  Unit[ size unit ]
  Range: (0, ∞)
- **MinSizeTask**
  Minimum size of a task within a sprint.
  Unit: [size unit]
  Range: $(0, \infty)$

- **MaxSizeTask**
  Maximum size of a task within a sprint.
  Unit: [size unit]
  Range: $(0, \infty)$

- **AverageVerEffectiveness**
  Average verification effectiveness for a task.
  Unit: [%]
  Range: $(0, 100]$ 

- **RetestCoverageSprint**
  Coverage of fault detection between sprints.
  Unit: [%]
  Range: $(0, 100]$

- **RetestCoverageTask**
  Coverage of fault detection between tasks.
  Unit: [%]
  Range: $(0, 100]$

- **QualityLimit**
  Quality Limit for a task (maximal faults / size).
  Unit: [# / size unit]
  Range: $(0, \infty)$

- **AverageFaultInjectionPerSizeUnit**
  Number of faults injected during the development phase.
  Unit: [# / size unit]
  Range: $[0, \infty)$

- **AverageFaultInjectionPerFaultRemoval**
  Number of faults injected during the rework phase.
  Unit: [# / #]
  Range: $[0, \infty)$

- **SeedTask**
  Seed of task random generator.
  Unit: [ ]
  Range: $(-\infty, \infty)$

- **SeedSprint**
  Seed of sprint random generator.
  Unit: [ ]
  Range: $(-\infty, \infty)$
6.2.5 Outputs

- **SizeDelivered**
  Rate of the size of the artifact delivered as a release. The rate is determined by the artifact size and the delivery productivity.
  Unit: [ size unit / time unit ]
  Range: [0, ∞)

- **SprintActive**
  Indicator, whether an sprint is in the active phase (value 1) or in the delivery phase (value 0).
  Unit: [ ]
  Range: {0, 1}

- **UndetectedFaultsLevel**
  Current number of undetected faults in the parts already finished.
  Unit: [ # ]
  Range: [0, ∞)

6.2.6 Generic Description

6.2.6.1 Top

The top level of the component consists of the subsystems *FaultAccumulation*, *Sprint*, and *ProductBacklog*. The latter is an instance of the component *WorkBreakDown*, see Sec. 5.5.

*ProductBacklog* inputs:

\[ \text{ProductBacklog.SizeRateWhole} = \text{SizeReceived} + \text{Sprint.SizeUnfinished} \]  \hspace{1cm} (6.21)

\[ \text{ProductBacklog.Reset} = \text{Reset} \]  \hspace{1cm} (6.22)

\[ \text{ProductBacklog.Next} = \text{SprintActive} \]  \hspace{1cm} (6.23)

*ProductBacklog* parameters:

\[ \text{ProductBacklog.MinSize} = \text{MinSizeSprint} \]  \hspace{1cm} (6.24)

\[ \text{ProductBacklog.MaxSize} = \text{MaxSizeSprint} \]  \hspace{1cm} (6.25)

\[ \text{ProductBacklog.Seed} = \text{SeedSprint} \]  \hspace{1cm} (6.26)

\[ \text{ProductBacklog.ConversionRate} = \frac{\text{MaxSizeSprint}}{\text{PlanningLength}} \]  \hspace{1cm} (6.27)
**FaultAccumulation** inputs:

\[
\begin{align*}
\text{FaultAccumulation.SizeForIteration} &= \text{ProductBacklog.SizeRatePart} \\
\text{FaultAccumulation.SizePart} &= \text{ProductBacklog.SizePart} \\
\text{FaultAccumulation.UndetectedFaultsDelivered} &= \text{Sprint.UndetectedFaultsDelivered} \\
\text{FaultAccumulation.Reset} &= \text{EnableSprint} \\
\text{FaultAccumulation.CycleActive} &= \text{SprintActive}
\end{align*}
\]

**Sprint** inputs:

\[
\begin{align*}
\text{Sprint.SizeReceived} &= \text{ProductBacklog.SizeRatePart} \\
\text{Sprint.UndetectedFaultsReceived} &= \text{ProductBacklog.FaultPropagationRate} \\
\text{Sprint.DevelopmentProductivity} &= \text{DevelopmentProductivity} \\
\text{Sprint.VerificationProductivity} &= \text{VerificationProductivity} \\
\text{Sprint.DeliveryProductivity} &= \text{DeliveryProductivity} \\
\text{Sprint.RemovalProductivity} &= \text{RemovalProductivity} \\
\text{Sprint.Enable} &= \text{EnableSprint}
\end{align*}
\]

**Scrum** outputs:

\[
\begin{align*}
\text{SizeDelivered} &= \text{Sprint.SizeDelivered} \\
\text{SprintActive} &= \text{Sprint.SprintActive} \\
\text{UndetectedFaultsLevel} &= \text{FaultAccumulation.UndetectedFaultsLevel}
\end{align*}
\]

### 6.2.6.2 Subsystem FaultAccumulation

The subsystem *FaultAccumulation* consists of the subsystems *HiddenFaults* and *PendingFaults*, which are instances of the component *LevelVariableResetable*, see Sec. 8.2, and a component *RateConversion*, see Sec. 8.4.

**HiddenFaults** inputs:

\[
\begin{align*}
\text{HiddenFaults.In} &= \text{UndetectedFaultsDelivered} \\
\text{HiddenFaults.Reset} &= \text{Reset} \\
\text{HiddenFaults.Initial} &= 0 \\
\text{HiddenFaults.Out} &= \text{FaultPropagationRate}
\end{align*}
\]
**PendingFaults** inputs:

\[
\begin{align*}
\text{PendingFaults.In} &= \text{FaultPropagationRate} \\
\text{PendingFaults.Reset} &= \text{CycleActive} \\
\text{PendingFaults.Initial} &= 0 \\
\text{PendingFaults.Out} &= \text{UndetectedFaultsDelivered}
\end{align*}
\]

**RateConversion** inputs:

\[
\begin{align*}
\text{RateConversion.AdjustedSize} &= \text{HiddenFaults.Level} \cdot \frac{\text{RetestCoverageSprint}}{100} \\
\text{RateConversion.BaseSize} &= \text{SizePart} \\
\text{RateConversion.BaseRate} &= \text{SizeForIteration}
\end{align*}
\]

**FaultAccumulation** outputs:

\[
\begin{align*}
\text{FaultPropagationRate} &= \text{RateConversion.AdjustedRate} \\
\text{UndetectedFaultsLevel} &= \text{HiddenFaults.Level} + \text{PendingFaults.Level}
\end{align*}
\]

### 6.2.6.3 Subsystem Sprint

The subsystem *Sprint* consists of the subsystems *ResourceManagement* and *IterationControl* and a component *WorkTestRework*, see Sec. 6.1

**ResourceManagement** inputs:

\[
\begin{align*}
\text{ResourceManagement.SizeDeliveredTask} &= \text{WorkTestRework.SizeDelivered} \\
\text{ResourceManagement.UndetectedFaultsFromTask} &= \text{WorkTestRework.UndetectedFaultsDelivered} \\
\text{ResourceManagement.SizeReceivedSprint} &= \text{SizeReceived} \\
\text{ResourceManagement.UndetectedFaultsSprint} &= \text{UndetectedFaultsReceived} \\
\text{ResourceManagement.Reset} &= \text{Enable} \\
\text{ResourceManagement.NextTask} &= \text{IterationControl.NextTask} \\
\text{ResourceManagement.Delivery} &= \text{IterationControl.Delivery} \\
\text{ResourceManagement.DeliveryProductivity} &= \text{DeliveryProductivity}
\end{align*}
\]

**IterationControl** inputs:

\[
\begin{align*}
\text{IterationControl.DevelopmentState} &= \text{WorkTestRework.StateIndex} \\
\text{IterationControl.Enable} &= \text{Enable} \\
\text{IterationControl.DeliveryFinished} &= \text{ResourceManagement.DeliveryFinished}
\end{align*}
\]
WorkTestRework inputs:

\[
\begin{align*}
\text{WorkTestRework.SizeReceived} &= \text{ResourceManagement.SizeToTask} \quad (6.67) \\
\text{WorkTestRework.UndetectedFaultsReceived} &= \text{ResourceManagement.UndetectedFaultsToTask} \quad (6.68) \\
\text{WorkTestRework.DetectedFaultsReceived} &= \text{WorkTestRework.DetectedFaultsDelivered} \quad (6.69) \\
\text{WorkTestRework.DevelopmentProductivity} &= \text{DevelopmentProductivity} \quad (6.70) \\
\text{WorkTestRework.VerificationProductivity} &= \text{VerificationProductivity} \quad (6.71) \\
\text{WorkTestRework.DeliveryProductivity} &= \text{DeliveryProductivity} \quad (6.72) \\
\text{WorkTestRework.RemovalProductivity} &= \text{RemovalProductivity} \quad (6.73) \\
\text{WorkTestRework.Reset} &= \text{IterationControl.DevelopmentReset} \quad (6.74)
\end{align*}
\]

Sprint outputs:

\[
\begin{align*}
\text{SizeDelivered} &= \text{ResourceManagement.SizeDeliveredSprint} \quad (6.75) \\
\text{SizeUnfinished} &= \text{ResourceManagement.SizeUnfinished} \quad (6.76) \\
\text{UndetectedFaultsDelivered} &= \text{ResourceManagement.UndetectedFaultsDelivered} \quad (6.77) \\
\text{SprintActive} &= \text{Control.SprintActive} \quad (6.78)
\end{align*}
\]

6.2.6.4 Subsystem ResourceManagement

The subsystem ResourceManagement consists of the subsystems FaultManagement, DeliveryManagement, and SprintBacklog. The latter is an instance of the component WorkBreakDown, see Sec. 5.5.

SprintBacklog inputs:

\[
\begin{align*}
\text{SprintBacklog.SizeRateWhole} &= \text{SizeReceivedSprint} + \text{DeliveryManagement.BacklogSizeRate} \quad (6.79) \\
\text{SprintBacklog.Reset} &= \text{Reset} \quad (6.80) \\
\text{SprintBacklog.Next} &= \text{NextTask} \quad (6.81)
\end{align*}
\]

SprintBacklog parameters:

\[
\begin{align*}
\text{SprintBacklog.MinSize} &= \text{MinSizeTask} \quad (6.82) \\
\text{SprintBacklog.MaxSize} &= \text{MaxSizeTask} \quad (6.83) \\
\text{SprintBacklog.Seed} &= \text{SeedTask} \quad (6.84) \\
\text{SprintBacklog.ConversionRate} &= \frac{10 \cdot \text{MaxSizeTask} \cdot \text{MaxSizeSprint}}{\text{SprintLength} \cdot \text{MinSizeTask}} \quad (6.85)
\end{align*}
\]
**FaultManagement inputs:**

\[
\text{FaultManagement.SizePartRate} = \text{ProductBacklog.SizeRatePart} \quad (6.86)
\]

\[
\text{FaultManagement.SizePart} = \text{ProductBacklog.SizePart} \quad (6.87)
\]

\[
\text{FaultManagement.UndetectedFaultsDeliveredTask} = \text{UndetectedFaultsFromTask} \quad (6.88)
\]

\[
\text{FaultManagement.UndetectedFaultsSprint} = \text{UndetectedFaultsSprint} \quad (6.89)
\]

\[
\text{FaultManagement.UndetectedFaultsUnfinished} = \text{DeliveryManagement.UndetectedFaultsDelivered} \quad (6.90)
\]

\[
\text{FaultManagement.Reset} = \text{Reset} \quad (6.91)
\]

**DeliveryManagement inputs:**

\[
\text{DeliveryManagement.FaultsPending} = \text{UndetectedFaultsToTask} \quad (6.92)
\]

\[
\text{DeliveryManagement.SizePending} = \text{SizeToTask} \quad (6.93)
\]

\[
\text{DeliveryManagement.BacklogSize} = \text{SprintBacklog.RemainingSize} \quad (6.94)
\]

\[
\text{DeliveryManagement.RemainingFaults} = \text{FaultManagement.RemainingFaults} \quad (6.95)
\]

\[
\text{DeliveryManagement.NextTask} = \text{NextTask} \quad (6.96)
\]

\[
\text{DeliveryManagement.Delivery} = \text{Delivery} \quad (6.97)
\]

\[
\text{DeliveryManagement.SizeDeliveredTask} = \text{SizeDeliveredTask} \quad (6.98)
\]

\[
\text{DeliveryManagement.Reset} = \text{Reset} \quad (6.99)
\]

\[
\text{DeliveryManagement.DeliveryProductivity} = \text{DeliveryProductivity} \quad (6.100)
\]

**ResourceManagement outputs:**

\[
\text{SizeDeliveredSprint} = \text{DeliveryManagement.SizeDeliveredSprint} \quad (6.101)
\]

\[
\text{SizeUnfinished} = \text{DeliveryManagement.SizeUnfinished} \quad (6.102)
\]

\[
\text{UndetectedFaultsDelivered} = \text{DeliveryManagement.UndetectedFaultsDelivered} \quad (6.103)
\]

\[
\text{SizeToTask} = \text{SprintBacklog.SizeRatePart} \quad (6.104)
\]

\[
\text{UndetectedFaultsToTask} = \text{FaultManagement.UndetectedFaultsReceivedTask} \quad (6.105)
\]

\[
\text{DeliveryFinished} = \text{DeliveryManagement.DeliveryFinished} \quad (6.106)
\]

**6.2.6.5 Subsystem IterationControl**

\[
\text{Delivery} = (\text{ElapsedTime} \geq \text{SprintLength}) \quad (6.107)
\]
\[ \text{DevelopmentReset} = (\text{Rising edge of Enable}) \lor \text{Delivery} \]  
\hspace{1cm} (6.108)

\[ \text{SprintActive} = \text{Enable} \land \neg \text{Delivery} \]  
\hspace{1cm} (6.109)

\[ \text{NextTask} = (\text{DevelopmentState} = 0) \land \text{SprintActive} \land \]  
\[ (\text{ElapsedTime} \geq \text{PlanningLength}) \]  
\hspace{1cm} (6.110)

\[ \frac{d}{dt} \text{ElapsedTime} = 1 \]  
\hspace{1cm} (6.111)

Reset : \( \text{ElapsedTime} = 0 \) when \( \text{RestartCondition} = 1 \)

\[ \text{RestartCondition} = (\text{DeliveryFinished} \land \text{Delivery}) \lor \neg \text{Enable} \]  
\hspace{1cm} (6.112)

**6.2.6.6 Subsystem FaultManagement**

The subsystem \textit{FaultAccumulation} consists of the subsystems \textit{UndetectedFaults}, which is an instance of the component \textit{LevelVariableResetable}, see Sec. 8.2, and a component \textit{RateConversion}, see Sec. 8.4.

\textit{UndetectedFaults} inputs:

\[ \text{UndetectedFaults.In} = \text{UndetectedFaultsSprint} + \]  
\[ \text{UndetectedFaultsDeliveredTask} \]  
\hspace{1cm} (6.113)

\[ \text{UndetectedFaults.Reset} = \text{Reset} \]  
\hspace{1cm} (6.114)

\[ \text{UndetectedFaults.Initial} = 0 \]  
\hspace{1cm} (6.115)

\[ \text{UndetectedFaults.Out} = \text{UndetectedFaultsUnfinished} + \]  
\[ \text{UndetectedFaultsReceivedTask} \]  
\hspace{1cm} (6.116)

\textit{RateConversion} inputs:

\[ \text{RateConversion.AdjustedSize} = \text{RemainingFaults} \cdot \frac{\text{RetestCoverageTask}}{100} \]  
\hspace{1cm} (6.117)

\[ \text{RateConversion.BaseSize} = \text{SizePart} \]  
\hspace{1cm} (6.118)

\[ \text{RateConversion.BaseRate} = \text{SizeRatePart} \]  
\hspace{1cm} (6.119)

\textit{FaultAccumulation} outputs:

\[ \text{UndetectedFaultsReceivedTask} = \text{RateConversion.AdjustedRate} \]  
\hspace{1cm} (6.120)

\[ \text{RemainingFaults} = \text{UndetectedFaults.Level} \]  
\hspace{1cm} (6.121)
### 6.2.6.7 Subsystem DeliveryManagement

The subsystem `DeliveryManagement` consists of the subsystems `ToDeliverSize`, `TaskPending`, `FaultsPendingLevel`, which are instances of the component `LevelVariableResetable`, see Sec. 8.2, and of the subsystems `RateBacklog`, `RateSizePending`, `RateRemainingFaults`, `RateFaultsPending`, which are instances of the component `RateConversion`, see Sec. 8.4

#### `ToDeliverSize` inputs:

\[
\begin{align*}
ToDeliverSize.In &= SizeDeliveredTask \\
ToDeliverSize.Reset &= Reset \\
ToDeliverSize.Initial &= 0 \\
ToDeliverSize.Out &= SizeDeliveredSprint
\end{align*}
\]

#### `TaskPending` inputs:

\[
\begin{align*}
TaskPending.In &= SizePending \\
TaskPending.Reset &= NextTask \\
TaskPending.Initial &= 0 \\
TaskPending.Out &= RateSizePending.AdjustedRate
\end{align*}
\]

#### `FaultsPendingLevel` inputs:

\[
\begin{align*}
FaultsPendingLevel.In &= FaultsPending \\
FaultsPendingLevel.Reset &= NextTask \\
FaultsPendingLevel.Initial &= 0 \\
FaultsPendingLevel.Out &= RateFaultsPending.AdjustedRate
\end{align*}
\]

#### `RateBacklog` inputs:

\[
\begin{align*}
RateBacklog.AdjustedSize &= BacklogSize \\
RateBacklog.BaseSize &= ToDeliverSize.Level \\
RateBacklog.BaseRate &= \begin{cases} 
DeliveryProductivity & \text{if Delivery} \\
0 & \text{otherwise}
\end{cases}
\end{align*}
\]

#### `RateSizePending` inputs:

\[
\begin{align*}
RateSizePending.AdjustedSize &= TaskPending.Level \\
RateSizePending.BaseSize &= ToDeliverSize.Level \\
RateSizePending.BaseRate &= \begin{cases} 
DeliveryProductivity & \text{if Delivery} \\
0 & \text{otherwise}
\end{cases}
\end{align*}
\]
RateRemainingFaults inputs:

\[ \text{RateRemainingFaults.AdjustedSize} = \text{RemainingFaults} \]  
\[ \text{RateRemainingFaults.BaseSize} = \text{ToDeliverSize.Level} \]  
\[ \text{RateRemainingFaults.BaseRate} = \begin{cases} \text{DeliveryProductivity} & \text{if Delivery} \\ 0 & \text{otherwise} \end{cases} \]

RateFaultsPending inputs:

\[ \text{RateFaultsPending.AdjustedSize} = \text{FaultsPendingLevel.Level} \]  
\[ \text{RateFaultsPending.BaseSize} = \text{ToDeliverSize.Level} \]  
\[ \text{RateFaultsPending.BaseRate} = \begin{cases} \text{DeliveryProductivity} & \text{if Delivery} \\ 0 & \text{otherwise} \end{cases} \]

DeliveryManagement outputs:

\[ \text{SizeDeliveredSprint} = \begin{cases} \text{DeliveryProductivity} & \text{if (ToDeliverSize > 0) \land Delivery} \\ 0 & \text{otherwise} \end{cases} \]  
\[ \text{DeliveryFinished} = (\text{ToDeliverSize.Level} \leq 0) \]  
\[ \text{SizeUnfinished} = \text{RateSizePending.AdjustedRate} + \text{BacklogSizeRate} \]  
\[ \text{BacklogSizeRate} = \text{RateBacklog.AdjustedRate} \]  
\[ \text{UndetectedFaultsUnfinished} = \text{RateRemainingFaults.AdjustedRate} + \text{RateFaultsPending.AdjustedRate} \]

6.2.7 Simulink Implementation

![Simulink Diagram]

Figure 6.9: Scrum implementation
Figure 6.10: FaultAccumulation implementation

Figure 6.11: Sprint implementation
Figure 6.12: ResourceManagement implementation

Figure 6.13: IterationControl implementation
Figure 6.14: FaultManagement implementation

Figure 6.15: DeliveryManagement implementation
7 Management Components

7.1 HumanResourceChain

7.1.1 Purpose

HumanResourceChain includes three levels of experience. The growth of an level depends on the promotion time and quitting rate. To fix the total headcount to a specified value the quitting rate is compensated with new hired personnel. The hiring process is delayed by a specified delay time. This block evaluates required hiring rate to reach desired headcount and demonstrates possible distributions of three experience levels. Optional it is possible to allocate different productivity values to the three experience groups. This values are used to evaluate the actual adjusted productivity dependent on the personnel composition.

The model is based on [4].

7.1.2 Component

![HumanResourceChain component]

Figure 7.1: HumanResourceChain component

7.1.3 Inputs

- **RequiredStaff**
  
  Number of required persons additional to actual headcount. To reduce actual headcount use negative values.
  
  Unit: [ # ]
  
  Range: \((-\infty, \infty)\)

- **RawProductivity**

  Baseline productivity. This input port is optional and appears only if the parameter
InfluenceProductivity is set to the option Internal.
Unit: [ size unit / time unit ]
Range: [0, ∞)

7.1.4 Parameters

- **InitJuniors**
  Number of persons with junior experience level at the beginning of the simulation.
  Unit: [ # ]
  Range: (−∞, ∞)

- **InitMidLevels**
  Number of persons with mid. level experience at the beginning of the simulation.
  Unit: [ # ]
  Range: (−∞, ∞)

- **InitSeniors**
  Number of persons with senior experience level at the beginning of the simulation.
  Unit: [ # ]
  Range: (−∞, ∞)

- **JuniorsRelativeQuittingRate**
  Juniors relative quitting rate.
  Unit: [ % ]
  Range: [0, 100]

- **MidLevelsQuittingRate**
  Mid. levels relative quitting rate.
  Unit: [ % ]
  Range: [0, 100]

- **SeniorsQuittingRate**
  Seniors relative quitting rate.
  Unit: [ % ]
  Range: [0, 100]

- **MidLevelsPromotionTime**
  Time needed to advance from junior level to mid. level.
  Unit: [ time unit ]
  Range: [0, ∞)

- **SeniorsPromotionTime**
  Time needed to advance from mid. level to senior level.
  Unit: [ time unit ]
  Range: [0, ∞)

- **HiringDelay**
  Time delay between recruitment and hiring.
  Unit: [ time unit ]
  Range: [0, ∞)
• **InfluenceProductivity**
  Juniors slows down and seniors speeds up the achievable productivity. This block
  provides optional the evaluation of the actual adjusted productivity dependent on
  the personnel composition. To select the evaluation choose the option *Internal*. 
  Option: *External, Internal*

• **RelProdJunior**
  Relative productivity of a junior (relative to *RawProductivity*)
  Unit: [ % ]
  Range: [0, 100]

• **RelProdMidLevel**
  Relative productivity of a mid. level (relative to *RawProductivity*)
  Unit: [ % ]
  Range: [0, 100]

• **RelProdSenior**
  Relative productivity of a senior (relative to *RawProductivity*)
  Unit: [ % ]
  Range: [0, 100]

### 7.1.5 Outputs

• **Seniors**
  Actual number of persons at senior level.
  Unit: [ # ]
  Range: [0, ∞)

• **MidLevels**
  Actual number of persons at mid. level.
  Unit: [ # ]
  Range: [0, ∞)

• **Juniors**
  Actual number of persons at junior level.
  Unit: [ # ]
  Range: [0, ∞)

• **Staff**
  Sum of seniors, mid levels and juniors.
  Unit: [#]
  Range: [0, ∞)

• **HiringRate**
  Required hiring rate to reach desired headcount. The value depends on quitting
  rate and number of new persons.
  Unit: [ # / time unit ]
  Range: [0, ∞)

• **AdjustedProductivity**
Productivity taking into respect the different experience levels. This output port is optional and appears only if the parameter \textit{InfluenceProductivity} is set to the option \textit{Internal}.

Unit: [ size unit / time unit ]
Range: \([0, \infty)\)

### 7.1.6 Generic Description

\[
\text{Juniors} = \int \text{NewStaff} - \text{PromotingToMidLevel} - \text{JuniorQuitting} \, dt \quad (7.1)
\]

\[
\text{MidLevels} = \int \text{PromotingToMidLevel} - \text{PromotingToSeniors} - \text{MidLevelQuitting} \, dt \quad (7.2)
\]

\[
\text{Seniors} = \int \text{PromotingToSeniors} - \text{SeniorQuitting} \, dt \quad (7.3)
\]

\[
\text{Staff} = \text{Juniors} + \text{MidLevels} + \text{Seniors} \quad (7.4)
\]

\[
\text{NewStaff} = \int \frac{\text{HiringRate} - \text{NewStaff}}{\text{HiringDelay}} \, dt \quad (7.5)
\]

\[
\text{HiringRate} = \text{RequiredStaff} + \text{JuniorQuitting} + \text{MidLevelQuitting} + \text{SeniorQuitting} - \text{Staff} \quad (7.6)
\]

\[
\text{PromotingToMidLevel} = \frac{\text{Juniors}}{\text{MidLevelPromotionTime}} \quad (7.7)
\]

\[
\text{PromotingToSenior} = \frac{\text{MidLevels}}{\text{SeniorPromotionTime}} \quad (7.8)
\]

\[
\text{JuniorQuitting} = \text{Juniors} \cdot \text{JuniorQuittingRate} \quad (7.9)
\]

\[
\text{MidLevelQuitting} = \text{MidLevels} \cdot \text{MidLevelQuittingRate} \quad (7.10)
\]
SeniorQuitting = Seniors \cdot SeniorQuittingRate \quad (7.11)

Optional the adjusted productivity is evaluated with following equations.

\[
\text{RelProdJuniors} = \frac{RelProdJunior \cdot Juniors}{100} \quad (7.12)
\]

\[
\text{RelProdMidLevels} = \frac{RelProdMidLevel \cdot MidLevels}{100} \quad (7.13)
\]

\[
\text{RelProdSeniors} = \frac{RelProdSenior \cdot Seniors}{100} \quad (7.14)
\]

\[
\text{RelProd} = \frac{RelProdJuniors + RelProdMidLevels + RelProdSeniors}{Staff} \quad (7.15)
\]

AdjustedProductivity = RelProd \cdot RawProductivity \quad (7.16)

### 7.1.7 Simulink Implementation

![Simulink diagram](image)

Figure 7.2: \textit{HumanResourceChain} implementation
7.2 CommunicationOverhead

7.2.1 Purpose

The component models the communication overhead, which increases with the number of persons involved.
The model is based on [4]

7.2.2 Component

![CommunicationOverhead component]

Figure 7.3: CommunicationOverhead component

7.2.3 Inputs

- RawProductivity
  Productivity without communication overhead.
  Unit: [ size unit / time unit ]
  Range: [0, \(\infty\)]

- NumberOfPersons
  Number of persons involved.
  Unit: [#]
  Range: \((-\infty, \infty)\)

7.2.4 Parameters

- RelativeCommunicationOverhead
  Relative communication overhead.
  Unit: [ 1 / #^2 ]
  Range: [0, \(\infty\)]
7.2.5 Outputs

- AdjustedProductivity
  Productivity taking into account the communication overhead.
  Unit: \([\text{sizeunit/timeunit}]\)
  Range: \([0, \infty)\)

7.2.6 Generic Description

\[
\text{Overhead} = \text{RelativeCommunicationOverhead} \cdot \text{NumberOfPersons} \cdot (\text{NumberOfPersons} - 1) \tag{7.17}
\]

\[
\text{AdjustedProductivity} = \frac{\text{RawProductivity}}{(1 + \text{Overhead})} \tag{7.18}
\]

7.2.7 Simulink Implementation

Implemented as embedded matlab function.
7.3 KnowledgeAcquisition

7.3.1 Purpose

The KnowledgeAcquisition component models the dynamics of learning and aging of new learned knowledge. This effect influences productivity in software development projects and can be used to simulate changes in state of the art knowledge. The model is based on [4].

7.3.2 Component

Figure 7.4: KnowledgeAcquisition component

7.3.3 Inputs

- **RawProductivity**
  Baseline productivity. This input port is optional and appears only, if the parameter InfluenceProductivity is set to the option Internal.
  Unit: [ size unit / time unit ]
  Range: [0, ∞)
- **EnableLearning**
  Enables the learning process. When the learning process is not active the aging process (characterized by the parameter ObsolescenceFactor) is enabled.
  Unit: [ ]
  Range: {0, 1}
- **Reset**
  An rising edge on this input port resets the knowledge level to InitialKnowledge.
  Unit: [ ]
  Range: {0, 1}
7.3.4 Parameters

- **InitialKnowledge**
  Initial knowledge level at reset.
  Unit: [%]
  Range: [0, 100]

- **LearningCurve**
  Selection of the respective learning curves.
  Option: Linear learning curve, Logistic learning curve

- **LearningRate**
  Constant learning rate of the linear learning curve. This parameter is available only, if the option Linear learning curve is selected.
  Unit: [% / time unit]
  Range: (0, \(\infty\))

- **MinLearningRate**
  The logistic learning curve requires a minimum learning rate to guarantee learning at an knowledge level of zero. This parameter is available only, if the option Logistic learning curve is selected.
  Unit: [% / time unit]
  Range: (0, \(\infty\))

- **LearningFactor**
  In some learning curve models, the actual knowledge level influence the learning rate. The logistic learning curve is generated by feed back of the knowledge level fraction specified by this parameter and use this value as an learning rate speed up factor.
  This parameter is available only, if the option Logistic learning curve is active.
  Unit: [ ]
  Range: (0, \(\infty\))

- **ObsolescenceFactor**
  The knowledge aging is modeled as an fractional loss of the actual knowledge level per time unit.
  Unit: [1 / time unit]
  Range: (0, \(\infty\))

- **InfluenceProductivity**
  An low state of the art knowledge level can influence the productivity in an software project. This block provides optional the evaluation of the actual adjusted productivity dependent on the actual knowledge level. To select the evaluation choose the option Internal.
  Option: External, Internal
7.3.5 Outputs

- **Knowledge**
  Actual knowledge level.
  Unit: [%]
  Range: [0, 100]

- **KnowledgeChange**
  Learning rate indicates the actual learning speed and the state of the knowledge block. Positive values represents the learning state and negative values represents the aging state.
  Unit: [% / time unit ]
  Range: (-∞, ∞)

- **AdjustedProductivity**
  Productivity taking into account the relative knowledge. This output port is optional and appears only, if the parameter InfluenceProductivity is set to the option Internal.
  Unit: [ size unit / time unit ]
  Range: [0, ∞)

7.3.6 Generic Description

Saturation function (used in 7.20):

\[
sat(l, x, u) = \begin{cases} 
  l & \text{if } x < l \\ 
  u & \text{if } x > u \\ 
  x & \text{otherwise}
\end{cases}
\] (7.19)

Knowledge change:

\[
Knowledge = \text{sat}(0, \text{InitialKnowledge} + \int_{\text{ResetEvent}}^{t} \text{KnowledgeChange} \, d\tau, 100)
\] (7.20)

*ResetEvent*: Rising edge of *Reset*

Linear learning curve:

\[
\text{KnowledgeChange} = \begin{cases} 
  \text{LearningRate} & \text{if } \text{EnableLearning} > 0 \\ 
  \text{Knowledge} \cdot \text{ObsolescenceFactor} & \text{otherwise}
\end{cases}
\] (7.21)

Logistic learning curve:

\[
\text{KnowledgeChange} = \begin{cases} 
  \text{LogisticLearning} & \text{if } \text{EnableLearning} > 0 \\ 
  \text{Knowledge} \cdot \text{ObsolescenceFactor} & \text{otherwise}
\end{cases}
\]
LogisticLearning = \text{MinLearningRate} + \text{Knowledge} \cdot \text{LearningFactor} \cdot (100 - \text{Knowledge}) \quad (7.23)

Optional:

\text{AdjustedProductivity} = \text{RawProductivity} \cdot \frac{\text{Knowledge}}{100} \quad (7.24)

7.3.7 Simulink Implementation

Figure 7.5: KnowledgeAcquisition implementation

Figure 7.6: LinearLearningCurve subsystem
Figure 7.7: LogisticLearningCurve subsystem
7.4 ExperienceAcquisition

7.4.1 Purpose

The ExperienceAcquisition component models productivity dependent on experience from actual or similar projects in the past. The productivity value grows log-linear with the actual experience value. The model is based on [4].

7.4.2 Component

![Diagram of ExperienceAcquisition component](image)

Figure 7.8: ExperienceAcquisition component

7.4.3 Inputs

- **ExperiencingRate**
  The experiencing rate is in general proportional to the work rate, i.e. the productivity. However, it might be smaller due to “work without gaining experience”.
  Unit: [ size unit / time unit ]
  Range: [0, ∞)

- **NumberOfPersons**
  The number of persons influencing the productivity and the experiencing rate.
  Unit: [#]
  Range: [0, ∞)

- **EnableExperiencing**
  Enables the experiencing process.
  Unit: [ ]
  Range: 0, 1

- **Reset**
  An rising edge on this input port resets the experience level from the actual project to zero. The experience from similar projects in the past remains.
  Unit: [ ]
  Range: 0, 1
7.4.4 Parameters

- **ExperienceCurve**
  
  Selection of different learning curves. The supported options differ only in the treatment of experience from similar projects in the past.
  
  Option: DeJong, S-Curve

- **LearningRate**
  
  The learning rate influences the experiencing speed. A higher learning rates slows down, a lower learning rates speeds up the experiencing rate.
  
  Unit: \([\%]\)
  
  Range: \([0, 100]\)

- **FirstUnitCost**
  
  Required effort for the first unit. In general the required effort for the next unit drops down.
  
  Unit: \(\text{[time unit} \cdot \#) / \text{size unit}\]
  
  Range: \((0, \infty)\)

- **MinUnitCost**
  
  Smallest achievable value for the required effort per unit. This value has to be smaller then the required effort for the first unit.
  
  Unit: \(\text{[(time unit} \cdot \#) / \text{size unit}\]
  
  Range: \((0, \infty)\)

- **TransferredExperience**
  
  Transferred experience from similar projects in the past. This parameter is available only, if the option *S-Curve* is active.
  
  Unit: \([\text{size unit}]\]
  
  Range: \([0, \infty)\)

7.4.5 Outputs

- **Experience**
  
  Is the actual experience level. This level depends on work done and transferred experience from similar done projects in the past.
  
  Unit: \([\text{size unit}]\]
  
  Range: \([0, \infty)\)

- **Productivity**
  
  Productivity taking into account the experience level.
  
  Unit: \([\text{size unit} / \text{time unit}]\]
  
  Range: \([0, \infty)\)

- **Productivity/Person**
  
  Productivity value divided by the allocated personnel.
  
  Unit: \([\text{size unit} / (\text{time unit} \cdot \#)]\]
  
  Range: \([0, \infty)\)
7.4.6 Generic Description

\[
Productivity = \frac{Productivity/Person \cdot NumberOfPersons}{PersonEffort} \quad (7.25)
\]

\[
Productivity/Person = \frac{1}{PersonEffort} \quad (7.26)
\]

\[
PersonEffort = \text{MinUnitCost} + (\text{FirstUnitCost} - \text{MinUnitCost}) \cdot (\text{Experience} + 1)^{b(\text{LearningRate}/100)} \quad (7.27)
\]

\[
\frac{d}{dt} \text{ExperienceLevel} = \text{ExperiencingRate} \quad (7.28)
\]

Reset: \: \text{ExperienceLevel} = 0 \text{ at } t = \text{ResetEvent}

\text{ResetEvent} : \text{Rising edge of Reset}

Selected learning curve: \text{DeJong}:

\[
\text{Experience} = \text{ExperienceLevel} \quad (7.29)
\]

\text{S-Curve}:

\[
\text{Experience} = \text{ExperienceLevel} + \text{TransferredExperience} \quad (7.30)
\]

7.4.7 Simulink Implementation

![Simulink Diagram](image)

Figure 7.9: \textit{ExperienceAcquisition} implementation
7.5 Training

7.5.1 Purpose

The block models the assimilation process, when new personnel is added to a project. Adding new people to a project leads to a decrease of the productivity, because experienced developers are needed for training.

The model is based on [4].

7.5.2 Component

![Training component diagram]

Figure 7.10: *Training* component

7.5.3 Inputs

- *AllocationRate*
  Rate of new personnel added per time step.
  Unit: \([ \# / \text{time unit} ]\)
  Range: \((-\infty, \infty)\)

7.5.4 Parameters

- *AssimilationRate*
  Relative time needed to assimilate new persons.
  Unit: \([ \text{time unit} ]\)
  Range: \((-\infty, \infty)\)

- *InitialPersonnel*
  Number of experienced persons at the beginning of the simulation.
  Unit: \([ \# / \text{time unit} ]\)
  Range: \((-\infty, \infty)\)
• *TrainingOverhead*
  Relative number of experienced persons needed for training for each new person.
  Unit: [ # / time unit ]
  Range: \((-\infty, \infty)\)

### 7.5.5 Outputs

- **Trained**
  Number of experienced persons available.
  Unit: [ # ]
  Range: \([0, \infty)\)

- **Untrained**
  Number of unexperienced persons available.
  Unit: [ # ]
  Range: \([0, \infty)\)

### 7.5.6 Generic Description

\[
\frac{d}{dt} Untrained = \text{AllocationRate} - \frac{Untrained}{\text{AssimilationRate}}
\]

Lower limit for *Untrained*: 0

\[
Trained\text{Brutto} = \text{InitialPersonnel} + \int_0^t \text{AssimilationRate} \cdot Untrained + Deallocation \, dt
\]

Lower limit for *TrainedBrutto*: 0

\[
Deallocation = \begin{cases} 
\text{AllocationRate} & \text{if } \text{AllocationRate} < 0 \land Untrained \leq 0 \\
0 & \text{otherwise}
\end{cases}
\]

\[
Trained = \max(Trained\text{Brutto} - \text{TrainingOverhead} \cdot Untrained, 0)
\]
7.5.7 Simulink Implementation

Figure 7.11: Training implementation
7.6 DelayBasedResourceAllocation

7.6.1 Purpose

This component models the allocation of additional resources in case of a pending delay of the end date. The starting point of such an intervention can be adjusted by the parameter *AwarenessThreshold*.

7.6.2 Component

![Diagram of DelayBasedResourceAllocation component](image)

Figure 7.12: *DelayBasedResourceAllocation* component

7.6.3 Inputs

- **SizeRequired**
  Required size of the task.
  Unit: [size unit]
  Range: \((0, \infty)\)

- **SizeProduced**
  Size already produced (i.e. already finished).
  Unit: [size unit]
  Range: \([0, \infty)\)

- **TimeSinceStart**
  Time elapsed since the start of the task.
  Unit: [time unit]
  Range: \([0, \infty)\)

- **PlannedDuration**
  Planned duration of the task.
  Unit: [time unit]
  Range: \((0, \infty)\)
• **MaximalAdditionalResources**
  Maximal additional resources, which can be allocated.
  Unit: [ resource unit ]
  Range: \([0, \infty)\)

### 7.6.4 Parameters

• **AwarenessThreshold**
  Part of the project receiving management awareness (percentage of total time prior to enddate).
  Unit: [ % ]
  Range: \([0, 100]\)

• **AcceptedDelay**
  Accepted delay (percentage deviation).
  Unit: [ % ]
  Range: \([0, 100]\)

• **AllocationRate**
  Possible allocation rate.
  Unit: [ resource unit / time unit ]
  Range: \([0, \infty)\)

### 7.6.5 Outputs

• **ManagedAllocationRate**
  Actual (managed) allocation rate.
  Unit: [ resource unit / time unit ]
  Range: \([0, \infty)\)

### 7.6.6 Generic Description

\[
\text{ManagedAllocationRate} = \begin{cases} 
\text{AllocationRate} & \text{if } \text{AllocationCondition} \\
0 & \text{otherwise}
\end{cases}
\]

\[
\text{AllocationCondition} = 
\text{Awareness} \land (\text{MaximalAdditionalResources} > \text{AddedResources}) 
\land \left( \frac{\text{SizeRequired} \cdot \text{TimeSinceStart}}{\text{SizeProduced} \cdot \text{PlannedDuration}} \geq (1 + \frac{\text{AcceptedDelay}}{100}) \right) 
\tag{7.35}
\]

\[
\text{Awareness} = \begin{cases} 
\text{true} & \text{if } \frac{\text{TimeSinceStart}}{\text{PlannedDuration}} \geq (1 - \frac{\text{AwarenessThreshold}}{100}) \\
\text{false} & \text{otherwise}
\end{cases}
\tag{7.36}
\]
\[ \frac{d}{dt} \text{AddedResources} = \text{ManagedAllocationRate} \]

Reset : \[ \text{AddedResources} = 0 \text{ if } \text{ResetCondition} \]

\[ \text{ResetCondition : } \neg \text{Awareness} \] (7.37)

### 7.6.7 Simulink Implementation

![Diagram 1](image1)

Figure 7.13: \textit{DelayBasedResourceAllocation} implementation

![Diagram 2](image2)

Figure 7.14: \textit{Allocation} subsystem
### 7.7 BacklogBasedResourceDistribution

#### 7.7.1 Purpose

This component distributes available resources among a set of tasks such that each task will be finished at the same time (ideally). The model is based on [4].

#### 7.7.2 Component

![BacklogBasedResourceDistribution component](image)

Figure 7.15: BacklogBasedResourceDistribution component

#### 7.7.3 Inputs

- **TotalResources**
  
  Available resources.
  
  Unit: [resource unit]
  
  Range: [0, ∞)

- **ToProduceSize**
  
  Vector of sizes (dimension n), which still need to be produced for each task (backlog).
  
  Unit: [size unit]
  
  Range: [0, ∞)

#### 7.7.4 Parameters

- **TaskProductivity**
  
  Vector of productivities (dimension n) of one resource unit for each task.
  
  Unit: [size unit / (time unit & resource unit)]
  
  Range: (0, ∞)
• *TimeStep*
  Time interval between consecutive reallocations.
  Unit: [ time unit ]
  Range: \((0, \infty)\)

7.7.5 Outputs

• *AllocatedResources*
  Vector of resources (dimension \(n\)), which are allocated for the respective task.
  Unit: [ resource unit ]
  Range: \([0, \infty)\)

• *AllocatedProductivity*
  Vector of allocated productivities (dimension \(n\)) for each task.
  Unit: [ size unit / time unit]
  Range: \([0, \infty)\)

7.7.6 Generic Description

\[
\text{TotalResourcesSampled}_k = \text{TotalResources}(k \cdot \text{TimeStep})
\]

\[
\text{ToProduceSizeSampled}_k = \text{ToProduceSize}(k \cdot \text{TimeStep})
\]

\[
\text{AllocationFactor}_k = \sum_{i=1}^{n} \frac{\text{ToProduceSizeSampled}_k[i]}{\text{TaskProductivity}[i]}
\]

\(n = \text{number of tasks}\)

\[
\text{fac} = \frac{\text{TotalResourcesSampled}_k}{\text{AllocationFactor}_k}
\]

\(\forall i \in 1, \ldots, n:\)

\[
\text{AllocatedProductivity}[i] = \begin{cases} 
\text{ToProduceSizeSampled}_k[i] \cdot \text{fac} & \text{if } \text{AllocationFactor}_k > 0 \\
0 & \text{otherwise}
\end{cases}
\]

\(\forall i \in 1, \ldots, n:\)

\[
\text{AllocatedResources}[i] = \frac{\text{AllocatedProductivity}[i]}{\text{TaskProductivity}[i]}
\]
7.7.7 Simulink Implementation

Figure 7.16: BacklogBasedResourceDistribution implementation
8 Utilities

8.1 LevelVariable

8.1.1 Purpose

The component models a level variable with input and output rates. The model is based on the system dynamics modelling paradigm (continuous simulation).

8.1.2 Component

![Diagram of LevelVariable component]

Figure 8.1: LevelVariable component

8.1.3 Inputs

- **In**
  - Input rate
  - Unit: [ size unit / time unit ]
  - Range: (−∞, ∞)
- **Out**
  - Output rate
  - Unit: [ size unit / time unit ]
  - Range: (−∞, ∞)

8.1.4 Parameters

None.
8.1.5 Outputs

- **Level**
  
  Current value of the level variable.
  
  Unit: [ size unit ]
  
  Range: \([0, \infty)\)

8.1.6 Generic Description

\[
\frac{d}{dt} \text{Level} = \text{In} - \text{Out}
\]  

(8.1)

Lower limit for **Level**: 0

8.1.7 Simulink Implementation

![Simulink Diagram](image)

Figure 8.2: **LevelVariable** implementation
8.2 LevelVariableResetable

8.2.1 Purpose

The component models a level variable with input and output rates. The level can be set to a specified value using a reset signal. The model is based on the system dynamics modelling paradigm (continuous simulation).

8.2.2 Component

![LevelVariableResetable component](image)

Figure 8.3: *LevelVariableResetable* component

8.2.3 Inputs

- **In**
  - Input rate
  - Unit: [ size unit / time unit ]
  - Range: \((-\infty, \infty)\)

- **Out**
  - Output rate
  - Unit: [ size unit / time unit ]
  - Range: \((-\infty, \infty)\)

- **InitialOut**
  - Value set at reset.
  - Unit: [ size unit / time unit ]
  - Range: \([0, \infty)\)

- **Reset**
  - Reset signal (reset at rising edge).
  - Unit: [ ]
  - Range: \({0, 1}\)
8.2.4 Parameters

None.

8.2.5 Outputs

- **Level**
  
  Current value of the level variable.
  
  Unit: [ size unit ]
  
  Range: \([0, \infty)\)

8.2.6 Generic Description

\[
\frac{d}{dt} Level = In - Out \tag{8.2}
\]

Reset: Level = Initial at \(t = ResetEvent\)

ResetEvent: Rising edge of Reset

Lower limit for Level: 0

8.2.7 Simulink Implementation

![Simulink Diagram](image)

Figure 8.4: `LevelVariableResetable` implementation
8.3 AsymmetricDelay

8.3.1 Purpose

In some cases the growth and attrition speeds of a signal are different. This block allows to define different time constants for growth and attrition.

The model is based on [4].

8.3.2 Component

![AsymmetricDelay component](image)

Figure 8.5: AsymmetricDelay component

8.3.3 Inputs

- **Input**
  
  Input signal which will be asymmetrically delayed in growth and attrition.
  
  Unit: [signalunit]
  
  Range: \((-\infty, \infty)\)

8.3.4 Parameters

- **InitialLevel**
  
  Initial signal level.
  
  Unit: [signalunit]
  
  Range: \((-\infty, \infty)\)

- **DelayGrowth**
  
  Delay time for increase.
  
  Unit: [TimeUnit]
  
  Range: \((0, \infty)\)

- **DelayAttrition**
  
  Delay time for decrease.
  
  Unit: [TimeUnit]
  
  Range: \((0, \infty)\)
8.3.5 Outputs

- **Output**
  Asymmetrically delayed output signal.
  Unit: \([\text{signalunit}]\)
  Range: \((-\infty, \infty)\)

8.3.6 Generic Description

\[
\frac{d}{dt} Output = \begin{cases}
  \text{Input} - \text{Output} & \text{if } \text{Input} - \text{Output} > 0 \\
  \frac{\text{DelayGrowth}}{\text{Input}-\text{Output}} & \text{otherwise}
\end{cases}
\]

\[Output(0) = \text{InitialLevel}\]  

(8.3)

8.3.7 Simulink Implementation

![AsymmetricDelay implementation]

Figure 8.6: AsymmetricDelay implementation
8.4 RateConversion

8.4.1 Purpose

This component synchronizes two rates.

8.4.2 Component

![Diagram of RateConversion component]

Figure 8.7: RateConversion component

8.4.3 Inputs

- *AdjustedSize*
  Size corresponding to the adjusted rate.
  Unit: [size unit]
  Range: \([0, \infty)\)

- *BaseSize*
  Size corresponding to the base rate.
  Unit: [size unit]
  Range: \([0, \infty)\)

- *BaseRate*
  Base rate, to which *AdjustedRate* is to be adjusted / synchronized.
  Unit: [size unit / time unit]
  Range: \([0, \infty)\)

8.4.4 Parameters

None.

8.4.5 Outputs

- *AdjustedRate*
  Adjusted / synchronized rate.
  Unit: [size unit / time unit]
  Range: \([0, \infty)\)
8.4.6 Generic Description

\[ \text{AdjustedRate} = \begin{cases} \frac{\text{BaseRate} \cdot \text{AdjustedSize}}{\text{BaseSize}} & \text{if BaseSize} > 0 \\ 0 & \text{otherwise} \end{cases} \]

8.4.7 Simulink Implementation

Implemented as embedded matlab function.
8.5 Size2Rate

8.5.1 Purpose

This component converts a size into a rate.

8.5.2 Component

![Size2Rate component](Figure 8.8: Size2Rate component)

8.5.3 Inputs

- **Size**
  Size value which should be converted into a rate.
  Unit: [size unit]
  Range: [0, ∞)

- **Reset**
  Reset which restarts the conversion.
  Unit: [
  Range: {0, 1}

8.5.4 Parameters

- **ConversionRate**
  Magnitude of the resulting rate.
  Unit: [size unit / time unit]
  Range: (0, ∞)

8.5.5 Outputs

- **Rate**
  Resulting rate.
  Unit: [size unit / time unit]
  Range: [0, ∞)
### 8.5.6 Generic Description

\[
\frac{d}{dt} \text{DeliveredSize} = \text{Rate} 
\]

Reset : \( \text{DeliveredSize} = 0 \) if ResetEvent

ResetEvent : Rising edge of Reset

\[
\text{Rate} = \begin{cases} 
\text{ConversionRate} & \text{if Size} > \text{DeliveredSize} \\
0 & \text{otherwise} 
\end{cases} 
\]

### 8.5.7 Simulink Implementation

![Simulink Diagram](image)

Figure 8.9: \( \text{Size2Rate} \) implementation
9 Examples

9.1 Effects of Different Prioritization Strategies

9.1.1 Model name

reqs_generator_example.mdl

9.1.2 SimSWE Components Used

RequirementsGenerator, Size2Rate, WorkWithExperience.

9.1.3 Purpose

The model demonstrates the use of the components Requirements Generator and WorkWithExperience and their respective interaction. It can be used to analyse the effects of different prioritization strategies.

9.1.4 Simulink Implementation

![Simulink Diagram](image_url)

Figure 9.1: reqs_generator_example.mdl
9.2 Brooks Law

9.2.1 Model name

brooks_law.mdl

9.2.2 SimSWE Components Used

DelayBasedResourceAllocation, Size2Rate, Training, WorkWithCommunicationOverhead.

9.2.3 Purpose

The model demonstrates the use of the components DelayBasedResourceAllocation, Training, and WorkWithCommunicationOverhead and their respective interaction. It can be used to analyse the effects of allocation strategies on project duration and delay.

9.2.4 Simulink Implementation

Figure 9.2: brooks_law.mdl
9.3 V-Models with and without Error Propagation

9.3.1 Model names

v_model.mdl
v_model_with_feedback_and_allocation.mdl

9.3.2 SimSWE Components Used

Scrum.

9.3.3 Purpose

The models demonstrate the use of the component *WorkTestRework* and its respective interaction.

*v_model.mdl* assumes that propagated errors are detected and corrected in the detecting phases without recursion into a previous phase.

*v_model_with_feedback_and_allocation.mdl* assumes that errors propagated from the development phases (analysis, design, or implementation) are detected only in the respective test phases. However, the correction of these errors is not part of the test phase, but fed back in the respective development phase as a correction task (with a size proportional to the number of detected errors). Moreover, the model can handle multiproject scenarios and includes an allocation model.

9.3.4 Simulink Implementation

![Simulink Diagram](image)

Figure 9.3: v_model.mdl
9.3.5 Simulink Implementation

Figure 9.4: v_model_with_feedback_and_allocation.mdl
9.4 Scrum process

9.4.1 Model names

scrum_example.mdl

9.4.2 SimSWE Components Used

Work-Test-Rework, Size2Rate.

9.4.3 Purpose

The models demonstrate the use of the component Scrum.

9.4.4 Simulink Implementation

Figure 9.5: scrum_example.mdl
Part II

SimSWE (extended)
10 Generators

10.1 TaskGeneratorD

10.1.1 Purpose

The TaskGeneratorD creates objects of type Task in random time intervals. Each object contains the attributes Size and Priority. Additionally a time stamp is added to the object to save simulation time of generation. The probability of occurrence and the values of the attributes are adjustable parameters.

10.1.2 Component

![TaskGeneratorD component](image)

Figure 10.1: TaskGeneratorD component

10.1.3 Inputs

None.

10.1.4 Parameters

- Distribution
  Probability distribution
  possible values: gaussian, uniform
- minCreateTaskTime
  Minimum time between the occurrence of two tasks
  Unit: [time unit]
  Range: [0, ∞)
- maxCreateTaskTime
  Maximum time between the occurrence of two tasks
Unit: [time unit]
Range: $[0, \infty)$

- **meanCreateTaskTime**
  Average time between the occurrence of two tasks
  Unit: [time unit]
  Range: $[0, \infty)$

- **varCreateTaskTime**
  Variance of time between the occurrence of two tasks
  Unit: [(time unit)$^2$]
  Range: $[0, \infty)$

- **meanTaskSP**
  Mean size of a task
  Unit: [size unit]
  Range: $[0, \infty)$

- **varTaskSP**
  Variance of size of a task
  Unit: [(size unit)$^2$]
  Range: $[0, \infty)$

### 10.1.5 Outputs

- **Task**
  Object of type *Task* containing the following attributes:

  1. **Size**
     Current size of the task
     Unit: [size unit]
     Range: $[0, \infty)$

  2. **Priority**
     Current priority of the task
     Unit: [ ]
     Range: $[0, 10]$

  3. **StartTime**
     Creation time of the task
     Unit: [time unit]
     Range: $[0, \infty)$

### 10.1.6 Generic Description

Under construction.
10.1.7 Simulink Implementation

Figure 10.2: TaskGeneratorD implementation
10.2 MultiSizeTaskGeneratorD

10.2.1 Purpose

The MultiSizeTaskGeneratorD creates large tasks and small tasks in random time intervals. Both are objects of type Task are created. Each object contains the attributes Size and Priority. Typically small tasks appear more often than large tasks and usually they have a higher priority. Additionally a time stamp is added to the object to save simulation time of generation. The probability of occurrence and the values of the attributes are adjustable parameters.

10.2.2 Component

![MultiSizeTaskGeneratorD component](image)

Figure 10.3: MultiSizeTaskGeneratorD component

10.2.3 Inputs

None.

10.2.4 Parameters

- **Distribution**
  Probability distribution
  possible values: gaussian, uniform

- **minCreateProjectTime**
  Minimum time between the occurrence of two large tasks
  Unit: [ time unit ]
  Range: [0, ∞)

- **maxCreateProjectTime**
  Maximum time between the occurrence of two large tasks
  Unit: [ time unit ]
  Range: [0, ∞)

- **meanCreateProjectTime**
  Mean time between the occurrence of two large tasks
- **varCreateProjectTime**
  Variance of time between the occurrence of two large tasks
  Unit: \( (\text{time unit})^2 \)
  Range: \([0, \infty)\)

- **meanProjectSP**
  Mean size of a large task
  Unit: \( \text{size unit} \)
  Range: \([0, \infty)\)

- **varProjectSP**
  Variance of size of a large task
  Unit: \( (\text{size unit})^2 \)
  Range: \([0, \infty)\)

- **minCreateTaskTime**
  Minimum time between the occurrence of two small tasks
  Unit: \( \text{time unit} \)
  Range: \([0, \infty)\)

- **maxCreateTaskTime**
  Maximum time between the occurrence of two small tasks
  Unit: \( \text{time unit} \)
  Range: \([0, \infty)\)

- **meanCreateTaskTime**
  Average time between the occurrence of two small tasks
  Unit: \( \text{time unit} \)
  Range: \([0, \infty)\)

- **varCreateTaskTime**
  Variance of time between the occurrence of two small tasks
  Unit: \( (\text{time unit})^2 \)
  Range: \([0, \infty)\)

- **meanTaskSP**
  Average size of a small task
  Unit: \( \text{size unit} \)
  Range: \([0, \infty)\)

- **varTaskSP**
  Variance of size of a small task
  Unit: \( (\text{size unit})^2 \)
  Range: \([0, \infty)\)
10.2.5 Outputs

- **Task**
  
  Object of type *Task* containing the following attributes:
  
  1. **Size**
     
     Current size of the task
     
     Unit: [ size unit ]
     
     Range: [0, ∞)
  
  2. **Priority**
     
     Current priority of the task
     
     Unit: [ ]
     
     Range: [0, 10]
  
  3. **StartTime**
     
     Creation time of the task
     
     Unit: [ time unit ]
     
     Range: [0, ∞)

10.2.6 Generic Description

Under construction.

10.2.7 Simulink Implementation

![Simulink Diagram](image)

Figure 10.4: *MultiSizeTaskGeneratorD* implementation
11 Estimators

No component yet.
12 Generic Activities

12.1 ProductionD

12.1.1 Purpose

The ProductionD component models the work on a given task (e.g. development) based on a productivity. While the productivity is a continuous signal, a software project or artifact is represented by a discrete task object. Each of these objects must have the attributes Size at the input and Size and Quality at the output. When an object is passed to the component, the time to fulfill the work is calculated as a function of Size, Quality and current Productivity. The Quality attribute is calculated as the sum of the initial quality and a random value.

12.1.2 Component

![ProductionD component](image)

Figure 12.1: ProductionD component

12.1.3 Inputs

- **Productivity**
  Productivity available for the task.
  Unit: [ size unit / time unit ]
  Range: (0, \(\infty\))

- **Task**
  Object of type Task containing the following attributes:

  1. **Size**
     Designated size of the task
     Unit: [ size unit ]
     Range: [0, \(\infty\)]
2. Quality
   Current Quality of the task
   Unit: [ quality unit ]
   Range: [0, 1]

12.1.4 Parameters

- MeanQuality
  Sets the average produced quality
  Unit: [ ]
  Range: [0, 1]

12.1.5 Outputs

- Result
  Object of type Task containing the following attributes:
  1. Size
     Current size of the task
     Unit: [#]
     Range: [0, ∞)
  2. Quality
     Current Quality of the task
     Unit: [ ]
     Range: [0, 1]

12.1.6 Generic Description

\[
\text{ProcessingTime} = (1 - \text{Task.Quality}) \cdot \frac{\text{Task.Size}}{\text{Productivity}}
\] (12.1)

\[
\text{Result.Quality} = \text{Task.Quality} + (1 - \text{Task.Quality}) \cdot \min(\max(\text{ProducedQuality}, 0), 1)
\] (12.2)

ProducedQuality is a normally (Gaussian) distributed random variable, with the properties

\[
E(\text{ProducedQuality}) = \text{MeanQuality}
\] (12.3)

\[
\text{Var}(\text{ProducedQuality}) = 0.1
\] (12.4)
12.1.7 Simulink Implementation

Figure 12.2: ProductionD implementation
12.2 VerificationD

12.2.1 Purpose

The VerificationD component models a verification and validation (V&V) process on a given artifact or task. The artifact or task is represented by a discrete object of type Task with the attributes Size and Quality. When an object is passed to the block, the time necessary to test the artifact is calculated as a function of Size and Productivity. When the calculated time is passed, the object is separated into two groups. If an artifact has achieved the required quality, it is sent to the Passed-Output port, if not it is sent to the output port Failed.

12.2.2 Component

![VerificationD component](image)

Figure 12.3: VerificationD component

12.2.3 Inputs

- **Productivity**
  Productivity available for the task.
  Unit: [size unit / time unit]
  Range: (0, \(\infty\))

- **Artifact**
  Object of type Task containing the following attributes:
  
  1. **Size**
     Size of the artifact
     Unit: [size unit]
     Range: [0, \(\infty\)]
  
  2. **Quality**
     Current Quality of the artifact
     Unit: [ ]
     Range: [0, 1]
12.2.4 Parameters

- **MinQuality**
  Required quality to pass verification
  Unit: [ ]
  Range: [0, 1]

12.2.5 Outputs

- **Failed**
  Object of type Task containing the following attributes:
    1. **Size**
       Current size of the task
       Unit: [ size unit ]
       Range: [0, ∞)
    2. **Quality**
       Current quality of the task
       Unit: [ ]
       Range: [0, 1]

- **Passed**
  Object of type Task containing the following attributes:
    1. **Size**
       Current size of the task
       Unit: [ size unit ]
       Range: [0, ∞)
    2. **Quality**
       Current Quality of the task
       Unit: [ ]
       Range: [0, 1]

12.2.6 Generic Description

\[
\text{ProcessingTime} = \frac{\text{Size}}{\text{Productivity}}
\]  

(12.5)

\[
\text{OutputPort} = \begin{cases} 
\text{Passed} & \text{if Quality > minQuality} \\
\text{Failed} & \text{otherwise}
\end{cases}
\]  

(12.6)
12.2.7 Simulink Implementation

Figure 12.4: \textit{VerificationD} implementation
13 Generic Processes

13.1 WorkTestReworkD

13.1.1 Purpose

The WorkTestRework component models the production of an artifact followed by an associated verification and validation step. The artifact or task is represented by a discrete object of type Task with the attributes Size and Quality. If the artifact does not yet meet the specified quality criterion, it is scheduled for rework. Rework and verification are cyclic traversed until the quality criterion is fulfilled. In this case, the artifact is released.

13.1.2 Component

Figure 13.1: WorkTestReworkD component

13.1.3 Inputs

- VerificationProd
  Productivity of the verification step.
  Unit: [ size unit / time unit ]
  Range: (0, ∞)
- DevelopmentProd
  Productivity of the production step.
  Unit: [ size unit / time unit ]
  Range: (0, ∞)
- EnableInput
  Enables the input queue.
Unit: [ ]
Range: \{0,1\}

- **Task**
  Object of type Task containing the following attributes:

  1. **Size**
     Designated size of the task
     Unit: [ size unit ]
     Range: \([0, \infty)\)

  2. **Quality**
     Current Quality of the task
     Unit: [ quality unit ]
     Range: \([0, 1]\)

13.1.4 Parameters

- **MeanQuality**
  Sets the average produced quality
  Unit: [ ]
  Range: \([0, 1]\)

- **MinQuality**
  Required quality to pass verification
  Unit: [ ]
  Range: \([0, 1]\)

- **maxWorkCount**
  Maximum number of tasks in queue (controls the alternation of work and rework tasks)
  Unit: [ # ]
  Range: \([0, \infty]\)

13.1.5 Outputs

- **Result**
  Object of type Task containing the following attributes:

  1. **Size**
     Current size of the task
     Unit: [ # ]
     Range: \([0, \infty]\)

  2. **Quality**
     Current Quality of the task
     Unit: [ ]
     Range: \([0, 1]\)
13.1.6 Generic Description

Under construction.

13.1.7 Simulink Implementation

Figure 13.2: *WorkTestReworkD* implementation

Figure 13.3: *Control* subsystem
14 Management Components

14.1 KnowledgeAcquisitionExtended

14.1.1 Purpose

The knowledge block demonstrates the dynamics of learning and aging of new learned knowledge. This effects influence productivity in software development projects and can be used to simulate changes in state of the art knowledge.

The extended version includes an knowledge level control to hold values in an specified range.

The model is based on [4].

14.1.2 Component

![KnowledgeAcquisitionExtended component](image)

Figure 14.1: KnowledgeAcquisitionExtended component

14.1.3 Inputs

- **EnableLearning**
  Enables the learning process. Until the learning process is active the aging process is disabled.
  This input port is optional and appears only if the parameter KnowledgeLevelControl is set to the option External.
  Unit: [#]
  Range: [0, 1]

- **ResetKnowledge**
  An rising edge on this input port resets the knowledge level to zero.
• **RawProductivity**  
  This input port is optional and appears only if the parameter `InfluenceProductivity` is set to the option `Internal`.  
  Unit: [#/TimeUnit]  
  Range: [0, ∞)

### 14.1.4 Parameters

- **InitialKnowledge**  
  Initials knowledge level at the beginning of the simulation.  
  Unit: [%]  
  Range: [0, 100]

- **LearningCurve**  
  Is an popup menu which is an selection between different learning curves.  
  Option: *Linear learning curve, Logistic learning curve*

- **LearningRate**  
  Linear learning curve is generated with an constant learning rate and have to be assigned by the user.  
  This parameter appears only when the option *Linear learning curve* is active.  
  Unit: [#/TimeUnit]  
  Range: (0, ∞)

- **MinimumLearningRate**  
  Especially the logistic learning curve requires a minimum learning rate to guarantee learning at an knowledge level of zero.  
  This parameter appears only when the option *Logistic learning curve* is active.  
  Unit: [#/TimeUnit]  
  Range: (0, ∞)

- **LearningFactor**  
  In some learning curve models the actual knowledge level influence the learning rate. For example the logistic learning curve is generated by feed back the knowledge level fractional and use this value as an learning rate speed up factor.  
  This parameter appears only when the option *Logistic learning curve* is active.  
  Unit: [#]  
  Range: (0, ∞)

- **ObsolescenceFactor**  
  The knowledge aging is modeled as an fractional loss of the actual knowledge level per time unit.
KnowledgeLevelControl
The extended version includes an knowledge level control to hold values in a
specified range. To enable the internal knowledge level control choose the option
Internal.
Option: External, Internal

KnowledgeLevelToStartLearning
At the specified knowledge level the control system set the signal EnableLearning
to active.
This parameter appears only when the parameter KnowledgeLevelControl is set to
Intern.
Unit: [%]
Range: [0, 100]

KnowledgeLevelToStopLearning
At the specified knowledge level the control system set the signal EnableLearning
to inactive.
This parameter appears only when the parameter KnowledgeLevelControl is set to
Intern.
Unit: [%]
Range: [0, 100]

InfluenceProductivity
An low state of the art knowledge level can influence the productivity in an software project. This block provides optional the evaluation of the actual adjusted
productivity dependent on the actual knowledge level. To select the evaluation
choose the option Internal.
Option: External, Internal

14.1.5 Outputs

RelativeKnowledge
Relative knowledge is the actual knowledge level.
Unit: [%]
Range: [0, 100]

LearningRate
Learning rate indicates the actual learning speed and the state of the knowledge
block. Positive values represents the learning state and negative values represents
the aging state.
Unit: [#]
Range: (-∞, ∞)
• *AdjustedProductivity*
  This output port is optional and appears only if the parameter *InfluenceProductivity* is set to the option *Internal*.
  Unit: [#/TimeUnit]
  Range: [0, ∞)

### 14.1.6 Generic Description

Generic description depends on the selected learning curve.

**Linear learning curve:**

\[
RelativeKnowledge =
\begin{cases}
\int SetLearningRate \, dt & \text{if } EnableLearning > 0 \\
\int RelativeKnowledge \cdot ObsolescenceFactor \, dt & \text{if } EnableLearning \leq 0
\end{cases}
\] (14.1)

**Logistic learning curve:**

\[
LearningRate = MinLearningRate + RelativeKnowledge \\
\cdot LearningFactor \cdot (100 - RelativeKnowledge)
\] (14.2)

\[
RelativeKnowledge =
\begin{cases}
\int LearningRate \, dt & \text{if } EnableLearning > 0 \\
\int RelativeKnowledge \cdot ObsolescenceFactor \, dt & \text{if } EnableLearning \leq 0
\end{cases}
\] (14.3)

Optional the adjusted productivity is evaluated with following equation.

\[
AdjustedProductivity = \frac{RawProductivity \cdot RelativeKnowledge}{100}
\] (14.4)
14.1.7 Simulink Implementation

Figure 14.2: KnowledgeAcquisitionExtended implementation: Top

Figure 14.3: KnowledgeAcquisitionExtended implementation: Knowledge level control
15 Utilities

No component yet.
Bibliography


