Methodology and Tools for Development and Verification for formal Requirements fUML Models and for Complex Software and Hardware Systems Architecture

Samonov A.V. <a.samonov@mail.ru>
Samonova G.N. <g.samonova@mail.ru>
Relevance of the problem

Software Projects Statistics from Standish Group FY2011–2015 within CHAOS database

<table>
<thead>
<tr>
<th>Year</th>
<th>Successful</th>
<th>Challenged</th>
<th>Failed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1994</td>
<td>16</td>
<td>53</td>
<td>31</td>
</tr>
<tr>
<td>1996</td>
<td>27</td>
<td>46</td>
<td>40</td>
</tr>
<tr>
<td>1998</td>
<td>26</td>
<td>46</td>
<td>28</td>
</tr>
<tr>
<td>2000</td>
<td>26</td>
<td>51</td>
<td>15</td>
</tr>
<tr>
<td>2002</td>
<td>34</td>
<td>53</td>
<td>18</td>
</tr>
<tr>
<td>2004</td>
<td>29</td>
<td>46</td>
<td>19</td>
</tr>
<tr>
<td>2006</td>
<td>35</td>
<td>44</td>
<td>24</td>
</tr>
<tr>
<td>2008</td>
<td>32</td>
<td>42</td>
<td>21</td>
</tr>
<tr>
<td>2010</td>
<td>37</td>
<td>49</td>
<td>22</td>
</tr>
<tr>
<td>2011</td>
<td>29</td>
<td>56</td>
<td>17</td>
</tr>
<tr>
<td>2012</td>
<td>27</td>
<td>50</td>
<td>19</td>
</tr>
<tr>
<td>2013</td>
<td>31</td>
<td>55</td>
<td>17</td>
</tr>
<tr>
<td>2014</td>
<td>28</td>
<td>52</td>
<td>15</td>
</tr>
</tbody>
</table>

Chaos report

The percentage of projects that were on budget from FY2011-2015 within the new CHAOS database.

The percentage of projects that were on time from FY2011-2015 within the new CHAOS database.

The percentage of projects that were on target from FY2011-2015 within the new CHAOS database.

TRADITIONAL RESOLUTION FOR ALL PROJECTS

<table>
<thead>
<tr>
<th>Year</th>
<th>Successful</th>
<th>Challenged</th>
<th>Failed</th>
</tr>
</thead>
<tbody>
<tr>
<td>2011</td>
<td>39%</td>
<td>39%</td>
<td>22%</td>
</tr>
<tr>
<td>2012</td>
<td>37%</td>
<td>46%</td>
<td>17%</td>
</tr>
<tr>
<td>2013</td>
<td>39%</td>
<td>40%</td>
<td>19%</td>
</tr>
<tr>
<td>2014</td>
<td>36%</td>
<td>47%</td>
<td>17%</td>
</tr>
<tr>
<td>2015</td>
<td>36%</td>
<td>45%</td>
<td>19%</td>
</tr>
</tbody>
</table>

The relative cost of error correction at different stages of the LC

<table>
<thead>
<tr>
<th>Stage</th>
<th>Cost Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements</td>
<td>1x</td>
</tr>
<tr>
<td>Design</td>
<td>5x</td>
</tr>
<tr>
<td>Coding</td>
<td>10x</td>
</tr>
<tr>
<td>Development</td>
<td>20x</td>
</tr>
<tr>
<td>Acceptance</td>
<td>50x</td>
</tr>
<tr>
<td>Operation</td>
<td>&gt;150x</td>
</tr>
</tbody>
</table>
Obstacles:
1) objective complexity of the problem of constructing a formal representation of the system requirements on the basis of their original informal presentation;
2) availability of a wide range of languages and tools proposed for building models of analysis, architecture and implementation of the system, in the absence of clear and specific rules and recommendations for their application;
1. To develop a unified conceptual, model, language and tool development and verification environment, we used: – modeling languages UML, fUML, OCL, ALF;
2. To formalize requirements: ontologies of the automated system, quality software, ReqIF, .xsd
3. As means for verification we have chosen: VM fUML, SPIN (Promela), Rodin (Event-B), SMT-Lib, Z3, CVC-4, ALT-ERGO.
4. Eclipse Modeling Framework, Graphical Editing Framework, Papyrus, Moka - development environment and tools: framework, libraries and tools implemented within the Eclipse project:
Step 1: develop requirements model
Step 1: develop requirements model.
Requirements metamodel: useCase&Class
Step 1: develop requirements model.
Algorithm for constructing a formal description of the requirements for the developed system

Algorithm for constructing a formal description of the requirements for the developed system
Step 1: Verification of requirements model to software

**Formal model of requirements**

- Class
  - attribute1 : Type
  - operation1(Type) : void
  - operation2() : Type

- iod
  - e1
  - init
  - result

**Virtual machine for execution fUML (ALF) models**

- Executor
  - public class Executor {
    - public ParameterValueList execute (Behavior behavior)
    - Execution execution = this.locus.false
    - ... execution execute ();
    - ...
  - }

- Locus
  - ExecutionFactory

**Formal verification**

- Event-B
  - Promela

**Simulation and Debug**

- Rodin EvenB
- SPIN
- CPN Tools

**Analyse Events**

- Event
  - timestamp : long
  - getTimestamp() : long

- TraceEvent
  - activityExecutionID : int
  - getactivityExecutionID() : int

**Techniques and tools**

- Next tasks for VM fUML
  - execute
  - next step
  - resume
Step 1: Virtual machine fUML
Step 2: develop the model of software architecture
Step 2: Verification the model of software architecture

### Formal model of software architecture

- **Class**
  - attribute1 : Type
  - operation1(Type) : void
  - operation2() : Type

### Virtual machine for fUML (ALF) model execution

- **Executor**
- **fUML (ALF)**

### Events Analysis

- **Event**
  - timestamp : long
  - getTimestamp() : long

- **TraceEvent**
  - activityExecutionID : int
  - getactivityExecutionID() : int

### Next tasks for fUML Virtual Machine

- > execute
  - > next step
  - > resume

### Formal verification

- **Promela**
- **Event-B**
Step 2. Validation and verification of design solutions

1. Checking consistency and unprofitability of fUML charts

2. Estimation of object-oriented quality metrics of the class diagram of the project (Metrics for Object Oriented Design, MOOD):
   - MHF (Method Hiding Factor)
   - AHF (Attribute Hiding Factor)
   - MIF (Method Inheritance Factor)
   - AIF (Attribute Inheritance Factor)
   - POF (Polymorphism Factor)
   - COF (Coupling Factor)

3. Verification of the completeness and correctness of functional requirements implementation through the implementation of the architectural model in the fUML VM environment

4. Verification of design solutions (architecture models) by means of SPIN, Rodin, etc
Step 3: Generation and verification of program code

```java
public NodeConn_rIDisconnectCfm (in portId : model::modelComponentView::DataTypes::Cello_Port,
   in serverConn_r : model::modelComponentView::DataTypes::Cello_RiServerConn)
{
    /* @inline("C++")
      "ENTER("NodeConn_rIDisconnectCfm."");"
    */

    if(serverConn_r.serverConnId == serverConnId && connHalf[serverConn_r.connHalf].portId == portId)
    {
        connHalf[serverConn_r.connHalf].respondState =
            model::modelComponentView::DataTypes::NCCdataTypes::RespondState[RRI_RESPONDED_CFM];

        model::modelComponentView::DataTypes::U8 secondHalf =
            NodeConn_getSecondHalf(serverConn_r.connHalf);

        if(connHalf[secondHalf].respondState !=
            model::modelComponentView::DataTypes::NCCdataTypes::RespondState[WATING_FOR_RESPONSE])
        {
            model::modelComponentView::DataTypes::NCCdataTypes::SciState x =
                SciHandler_RL.SciHandler_getSciConnectionState();

            if(x == model::modelComponentView::DataTypes::NCCdataTypes::SciState[SCI_ATTACHED])
            {
                NodeConn_setDisconnectingSpasState();
                SciHandler_RL.SciHandler_sciReleaseConnReq(spasConnId, serverConnId);
                // Simulate NodeConn_sciReleaseConnCfm();
        
```
Step 3: Generation and verification of program code.

Methods program code verification
Implementation Stages of software-controlled process of software and hardware systems development and verification

1. Develop XML-scheme for technical specifications. Input it in text editor content controls
2. Develop structured technical specifications in text editor environment
3. Generate use case diagram, block(class) diagram

1. Build preliminary formal model of requirements: complex UC-diagram and Class-diagram model
2. Develop a formal model of requirements: $d_{uc}, d_{io}, d_{class}, d_{regs}$
3. Develop an architecture of automated systems and Software: $d_{act}, d_{seq}, d_{sm}, d_{class}, d_{regs}$
4. Develop tests for verification of the analysis model

Verify the model of requirements

Verifying the model of requirements

3. Develop an architecture of automated systems and Software: $d_{act}, d_{seq}, d_{sm}, d_{class}, d_{regs}$
4. Develop tests for verification of the complex software and hardware systems architecture

Verifying the architecture (project) of AS and Software

4.1. Build a control flow graph (CFG)
4.2. Convert the CFG to the notation ALF
4.3. Generate a test case for the analysis model
4.4. Generate a test case for architecture model verification
4.5. Generate a test case for the implementation verification

Develop the implementation

Verify the implementation
Concise Summary

Models and algorithms are developed

1. metamodel for formalization the requirements to software: UseCase, Interaction overview, class, requirements diagrams

2. metamodel for formalized description architecture software: fUML diagrams class, activity, sequence, state machine;

3. algorithms for development of formal models of requirements and architecture software;

4. algorithms for verification of models of requirements and architecture software by means of their modeling and analysis in the virtual fUML machine and execution of test scenarios built with the help of SMT / SAT solvers;

5. methodological recommendations for the implementation of the software-controlled process of development and verification software, using the developed models and algorithms.