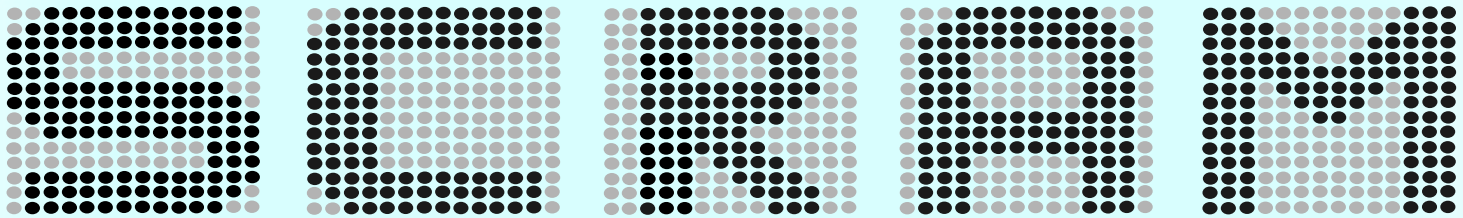

Command-line Risk Analysis Multi-tool

Olzhas Rakhimov <ol.rakhimov@gmail.com>

SCRAM 0.11.4

- Free, *as in free speech*, and open-source software (FOSS) project
  - GPLv3+ license
  - Only FOSS dependencies
  - Modern FOSS languages, libraries, and tools
  - Open standards (Open-PSA MEF, XML, RelaxNG, reST)
  - Modern software development best practices
- Debian Science package
- OIN licensee (a royalty-free-patent community)
- Available for all major platforms (GNU/Linux, Mac, Windows)
- Documentation website: http://scram-pra.org
- Development on GitHub: https://github.com/rakhimov/scram/

History

- Started as a student project in 2014
- Became a personal pet project
- Has been a playground to learn and practice software development
- 2300+ commits
- 20 releases
- 6+ years of effort (OpenHUB estimate with COCOMO model)

Development

- C++14 (~10 kSLOC / ~85%)
  - Compilers: GCC 4.9, Clang 3.4, Intel® 17.0.1
  - Dependencies: CMake, Boost, LibXML++, Qt 5, TCMalloc or JEMalloc
- Python (~2 kSLOC / ~15%)
  - 2.7
  - 3.3+
- Documentation Driven Development
  - The documentation source text in reStructuredText format
  - C++ documentation with Doxygen (100% coverage)
  - 34% comment/documentation lines (vs. 22% FOSS average)
- Test Driven Development (95% coverage)
- Design by Contract (no exceptions in **analysis code**)
- Policy-based design
- Defensive programming
- Analysis validation with XFTA

Quality Assurance

1. Source code management and version control with Git
2. Consistent coding styles based on Google C++ and Python Style Guides
3. Automated tests with GoogleTest and Nose
4. Continuous integration with Travis CI and AppVeyor
5. Automated documentation generation with Doxygen and Sphinx

C++

1. Performance profiling with `Gprof`, Valgrind, `perf`, Intel® VTune and Advisor
2. Code coverage check with Gcov and reporting with Coveralls
3. Memory management bug and leak detection with Valgrind
4. Static code analysis with Coverity and CppCheck
5. Cyclomatic complexity analysis with Lizard
6. Google style conformance check with Cpplint
7. Common C++ code problem check with cppclean
8. Consistent code formatting with ClangFormat
9. Component dependency analysis with cppdep

Quality Assurance (cont.)

Python

1. Code quality and style check with Pylint
2. Profiling with PyVmMonitor
3. Code coverage check with coverage and reporting with Codecov
4. Continuous code quality control on Landscape with Prospector

Targets

Metric	Before Release	On Release
C++ Code Coverage	80%	95%
C++ Defect Density	0.5 per 1000 SLOC	0.35 per 1000 SLOC
CCN	15	15
Python Code Coverage	80%	95%
Pylint Score	9.0	9.5
Documentation	Full	Full

Implemented Features in SCRAM 0.11.4

1. Static fault tree analysis
    - MOCUS
    - BDD (default)
    - ZBDD
2. Non-coherent model analysis
    - Minimal Cut Sets
    - Prime Implicants
3. Analysis with common-cause failure models
4. Probability calculations with importance analysis
5. Uncertainty analysis with Monte Carlo simulations
6. Fault tree generator
7. The shorthand format to the MEF converter

Performance

1. Prefer code quality, clarity, simplicity, elegance over performance
2. Trade memory for speed
3. Avoid approximations if possible

Baobab 1

No cut-off, all 46,188 MCS.

	MOCUS	ZBDD	BDD
Time, s	0.35	0.16	0.10
Memory, MiB	23	25	23

CEA9601

BDD			
Cut-off order	4	5	6
MCS	54,436	1,615,876	9,323,572
Time, s	1.6	3.4	12.6
Memory, MiB	215	310	1,350

System specs: Core i7-2820QM, Ubuntu 16.04 x64, GCC 5.4.0, Boost 1.58, TCMalloc 2.4

Open-PSA MEF in SCRAM 0.11.4

1. Label and Attributes
2. Public and Private Roles
3. Fault Tree Layer
    - Components
    - Basic events
    - House events
    - Gates (*nested formulae*)
4. Model Data
5. Common Cause Failure Groups (beta-factor, MGL, alpha-factor, phi-factor)
6. Parameters
7. Expressions
    - Constant expressions, System mission time, Parameter
    - Random deviate (normal, log-normal, histogram, uniform, gamma, beta)
    - Built-in expressions (exponential with 2 or 4 parameters, Weibull)
    - Arithmetic expressions

Challenges with MEF 2.0d

1. Minor errors in the MEF specification, the BNF or DTD schema
2. The location of the Model Data
3. Graphical representations for Cardinality, Imply, IFF gates
4. Graphical representations for **nested formulae**
5. The **include** feature
    - XML snippet valid by itself or within the context?
    - Problems with automatic validation with the schema
    - XInclude hack
    - Multiple input file processing as an alternative
6. Unspecified constraints on the name and reference formats
    - Problems with porting input files from one software to another

INHIBIT gate

```
<define-gate name="Gate">
  <attributes>
    <attribute name="flavor" value="inhibit"/>
  </attributes>
  <and>
    <event name="ConditionalEvent"/>
    <!-- argument events ... -->
  </and>
</define-gate>
```

CONDITIONAL event

```
<define-basic-event name="ConditionalEvent">
  <attributes>
    <attribute name="flavor" value="conditional"/>
  </attributes>
  <float value="0.4"/>
</define-basic-event>
```

UNDEVELOPED event

```
<define-basic-event name="Undeveloped">
  <attributes>
    <attribute name="flavor" value="undeveloped"/>
  </attributes>
  <float value="0.5"/>
</define-basic-event>
```

Challenges (cont.)

atleast gate

1. Many other names: Vote, Voting, Voting-OR, Combination, Combo, K/N, OR-MORE
2. *atmost*, OR-LESS, cardinality(1, k)

$$@^{\leqslant}(k, [x_i]) = @^{\geqslant}(n-k, [x'_i]) \ \& \ OR(x_i)$$

3. API: Atleast vs. AtLeast, atleast vs. at_least, ATLEAST vs. AT_LEAST

XML MEF report file size

- ~50x compression with `gzip`
- Reading with SAX parsers
- HDF5 or SQL database as an alternative
- Some binary format based on ZBDD serialization (probably, the most space efficient)

CEA9601 Report			
Cut-off order	4	5	6
MCS	54,436	1,615,876	9,323,572
Reporting, s	< 0.05	2.6	17.5
XML size, MB	9.3	329	2,200

Report CCF events in products

```
<results>
  <sum-of-products name="TopEvent" basic-events="6" products="6">
    <product order="2">
      <ccf-event ccf-group="Pumps" order="1" group-size="2">
        <basic-event name="PumpTwo"/>
      </ccf-event>
      <!-- ... -->
    </product>
  <!-- ... -->
  </sum-of-products>
</results>
```

Report importance factors

```
<results>
  <importance name="TopEvent" basic-events="4">
    <basic-event name="Pump" MIF="0.4" CIF="0.4" DIF="0.8" RAW="1.2" RRW="1.7"/>
    <basic-event name="Valve" MIF="0.4" CIF="0.4" DIF="0.8" RAW="1.2" RRW="1.7"/>
    <!-- ... -->
  </importance>
</results>
```

Proposals to the Open-PSA MEF

Host the MEF standard on GitHub

For the Open-PSA

1. 

rganization: https://github.com/open-psa/
2. Easy collaboration (more volunteers!)
3. Issue tracking
4. Free web-site hosting
5. Many more free perks for the project

For the Community

1. SCRAM and other FOSS projects as test-beds and early feedback for MEF features
2. Scripts to convert inputs from other formats to the MEF
3. Move the validation schemas from SCRAM to the MEF public repository
4. Provide validation input (fault tree, event tree, etc.) for implementers

Extra

1. Mailing lists for discussions (e.g., Google groups)

Specification for the Name format

1. Case-sensitive or case-agnostic (simplifies code for l10n/i18n)
2. Insensitive to leading and trailing whitespace characters (trim)
3. Consistent with XML NCName datatype
    - The first character must be alphabetic.
    - May contain alphanumeric characters and special characters like `_`, `-`.
    - No whitespace or other special characters like `:`, `,`, `/`, etc.
4. No double (or more) dashes `--`
5. No trailing dash
6. No periods `.`
    - Reserved for the Reference format, i.e., `fault_tree.component.event`

```
<define name="Identifier">
  <data type="NCName">
    <param name="pattern">[^\-.]+(-[^\-.]+)*</param>
  </data>
</define>
```

RelaxNG Schema

- Simpler than XSD
- Automated conversion to XSD with trang
- It's ready *for 2.0d*: MEF RelaxNG Schema, MEF RelaxNG Compact Schema

RelaxNG Compact (looks like the BNF!)

```
gate-definition =
  element define-gate { name, role?, label?, attributes?, formula }
```

RelaxNG

```
<define name="gate-definition">
  <element name="define-gate">
    <ref name="name"/>
    <optional> <ref name="role"/> </optional>
    <optional> <ref name="label"/> </optional>
    <optional> <ref name="attributes"/> </optional>
    <ref name="formula"/>
  </element>
</define>
```

```
reStructuredText
================

#. **Powerful** *markup* language
    - documentation
    - website
    - publishing
#. Simpler than ``LaTeX``
#. Supports inline ``LaTeX``, ``html``, image, code ... with Sphinx_

    .. math:: \frac{ \sum_{t=0}^{N}f(t,k) }{N}

#. Automated conversion to ``html``, ``LaTeX``, ``pdf``, ...
#. Automatic generation of the MEF documentation website

+--------------+--------+
| Table        | Head   |
+==============+========+
| Data         | entry  |
+--------------+--------+

.. _Sphinx: http://sphinx-doc.org/
```

reStructuredText

1. **Powerful** *markup* language
   - documentation
   - website
   - publishing
2. Simpler than `LaTeX`
3. Supports inline `LaTeX`, `html`, image, code ... with Sphinx

$$\frac{\sum_{t=0}^{N} f(t,k)}{N}$$

4. Automated conversion to `html`, `LaTeX`, `pdf`, ...
5. Automatic generation of the MEF documentation website

Table	head
Data	entry

Oops!

Floating point number format

XML Schema Part 2: Datatypes Second Edition, Lexical representation (§3.2.3.1)

> **decimal** has a lexical representation consisting of a finite-length sequence of decimal digits (#x30-#x39) separated by a **period** as a decimal indicator. An optional leading sign is allowed. If the sign is omitted, “+” is assumed. Leading and trailing zeroes are optional. If the fractional part is zero, the period and following zero(es) can be omitted. For example: -1.23, 12678967.543233, +100000.00, 210.

“Nothing is worse than having an itch you can never scratch.”

— Leon Kowalski, *Blade Runner*