Software Process in Geant4

Gabriele Cosmo
CERN IT/API-SI
Gabriele.Cosmo@cern.ch
Outline

- Overview on Software Processes
- The area of application
- Life-cycle processes in Geant4
- Software Process Improvement
  - Future evolutions
- Conclusions
Definitions...

- **Software Process**
  - A set of interrelated activities, which transform inputs into outputs (**ISO 12207/8402**)
    - used by an organisation or project to plan, manage, execute, monitor, control and improve any software related activity
  - Life-cycle processes are structured in *dimensions*:
    - Primary processes
      - includes all major functions of software development
    - Supporting processes
      - for supporting other processes with a purpose
    - Organisational processes
      - for corporate level management and improvement
Process Architecture

**Customer-Supplier**
CUS.1 Acquisition
  - CUS.1.1 Acquisition Preparation
  - CUS.1.2 Supplier Selection
  - CUS.1.3 Supplier Monitoring
  - CUS.1.4 Customer Acceptance
CUS.2 Supply
CUS.3 Requirements Elicitation (*)
CUS.4 Operation
  - CUS.4.1 Operational Use
  - CUS.4.2 Customer Support (*)

**Engineering**
ENG.1 Development
  - ENG.1.1 System Requirements A&D
  - ENG.1.2 Software Requirements Analysis
  - ENG.1.3 Software Design (*)
  - ENG.1.4 Software Construction (*)
  - ENG.1.5 Software Integration
  - ENG.1.6 Software Testing
  - ENG.1.7 System Integration & Testing (*)
ENG.2 System & Software Maintenance (*)

**Support**
SUP.1 Documentation (*)
SUP.2 Configuration Management (*)
SUP.3 Quality Assurance
SUP.4 Verification
SUP.5 Validation
SUP.6 Joint Reviews
SUP.7 Audit
SUP.8 Problem Resolution

**Management**
MAN.1 Management
MAN.2 Project Management
MAN.3 Quality Management
MAN.4 Risk Management

**Organisation**
ORG.1 Organisational Alignment
ORG.2 Improvement
  - ORG.2.1 Process Establishment
  - ORG.2.2 Process Assessment
  - ORG.2.3 Process Improvement (*)
ORG.3 Human Resource Management
ORG.4 Infrastructure
ORG.5 Measurement
ORG.6 Reuse
The area of application: Geant4

- More than 1200 classes distributed in 17 Categories
  - software components in the Booch terminology
  - complex Categories organised in a hierarchical structure
- Decomposition to domain Categories derived from the design Category diagram
  - one development team associated to one Category domain
The area of application: Geant4

- Development teams distributed world-wide
  - domain decomposition <> geographical location of teams
  - centralized coordination of domain Categories
  - local coordination of each Working Group
    - assignment of responsibilities and support
  - distributed resources and funds in a *dynamic* environment

- Coordination for a coherent development
  - computing environment, methods and tools
Requirements Elicitation

- General User Requirements (UR) collected during the R&D phase of the project (RD44)
  - GEANT3 user community involved
  - URD generated according to the ESA PSS-05 software engineering standard
  - regular update and versioning of the URD along the development process

- Change-management based on CVS
  - general URD currently under revision
  - maintenance and tracking of specific detailed URDs under responsibility of WG coordinators

- New requirements approval: by the TSB
  - ongoing process improvement
Software Design

- Adoption of the Booch methodology for OOAD since the R&D project start
  - chosen after deep evaluation of the existing methodologies (‘94)
  - tailored to project specific needs
  - supported by CASE tools (Rational-Rose)
  - UML notation adopted for design documents
    - Category diagrams, Class diagrams, Scenario diagrams, Class specifications
    - ongoing process improvement

- Software development structured in macro and micro processes showed very effective
  - iterative & incremental approach (spiral model)
  - loose domain coupling led to efficient WG structure
Software Construction

- Software packaging reflects the domain decomposition in Categories
  - Packaging of Categories and sub-Categories in relation to definition of abstract and concrete interfaces (*frameworks*)
    - Provide a set of services in a *re-usable* way
    - Software *toolkit* approach

- Essential and flexible guidelines for coding

- Code filtering with specialised tools
  - *Code Wizard*
  - Both in the *global* and *unit* context
    - Tool accessible from Web
System Testing

- Activity deployed to a specialised team (STT)
  - based on defined procedures
    - CVS tagging policy
    - automated through Web tools and scripts
      - Bonsai, LXR, Tinderbox
      - ongoing process improvement
  - test applications used also for system integration
    - run & tested on every supported platform/compiler
    - ongoing process improvement
  - user example applications used for acceptance

- Category tags submitted to testing in sequence according to the dependency flow dictated by the design category diagram

- Close collaboration with the release manager
<table>
<thead>
<tr>
<th>CAS</th>
<th>Description</th>
<th>Name/Institution</th>
<th>Location</th>
<th>Provider</th>
<th>Undergraduate</th>
<th>Graduate</th>
<th>PhD</th>
<th>Journal</th>
<th>ID</th>
<th>Start Date</th>
<th>End Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>00-70-0</td>
<td>CAL-TM-0070-0001-0001</td>
<td>Caltech</td>
<td>California</td>
<td>University of California</td>
<td>0-12</td>
<td>12-15</td>
<td>15-20</td>
<td>Journal of Chemical Society</td>
<td>00-70-0</td>
<td>12-23-2022</td>
<td>12-31-2023</td>
</tr>
<tr>
<td>00-70-0</td>
<td>CAL-TM-0070-0001-0002</td>
<td>Caltech</td>
<td>California</td>
<td>University of California</td>
<td>0-12</td>
<td>12-15</td>
<td>15-20</td>
<td>Journal of Chemical Society</td>
<td>00-70-0</td>
<td>12-23-2022</td>
<td>12-31-2023</td>
</tr>
<tr>
<td>00-70-0</td>
<td>CAL-TM-0070-0001-0003</td>
<td>Caltech</td>
<td>California</td>
<td>University of California</td>
<td>0-12</td>
<td>12-15</td>
<td>15-20</td>
<td>Journal of Chemical Society</td>
<td>00-70-0</td>
<td>12-23-2022</td>
<td>12-31-2023</td>
</tr>
<tr>
<td>00-70-0</td>
<td>CAL-TM-0070-0001-0004</td>
<td>Caltech</td>
<td>California</td>
<td>University of California</td>
<td>0-12</td>
<td>12-15</td>
<td>15-20</td>
<td>Journal of Chemical Society</td>
<td>00-70-0</td>
<td>12-23-2022</td>
<td>12-31-2023</td>
</tr>
<tr>
<td>00-70-0</td>
<td>CAL-TM-0070-0001-0005</td>
<td>Caltech</td>
<td>California</td>
<td>University of California</td>
<td>0-12</td>
<td>12-15</td>
<td>15-20</td>
<td>Journal of Chemical Society</td>
<td>00-70-0</td>
<td>12-23-2022</td>
<td>12-31-2023</td>
</tr>
<tr>
<td>00-70-0</td>
<td>CAL-TM-0070-0001-0006</td>
<td>Caltech</td>
<td>California</td>
<td>University of California</td>
<td>0-12</td>
<td>12-15</td>
<td>15-20</td>
<td>Journal of Chemical Society</td>
<td>00-70-0</td>
<td>12-23-2022</td>
<td>12-31-2023</td>
</tr>
<tr>
<td>00-70-0</td>
<td>CAL-TM-0070-0001-0007</td>
<td>Caltech</td>
<td>California</td>
<td>University of California</td>
<td>0-12</td>
<td>12-15</td>
<td>15-20</td>
<td>Journal of Chemical Society</td>
<td>00-70-0</td>
<td>12-23-2022</td>
<td>12-31-2023</td>
</tr>
<tr>
<td>00-70-0</td>
<td>CAL-TM-0070-0001-0008</td>
<td>Caltech</td>
<td>California</td>
<td>University of California</td>
<td>0-12</td>
<td>12-15</td>
<td>15-20</td>
<td>Journal of Chemical Society</td>
<td>00-70-0</td>
<td>12-23-2022</td>
<td>12-31-2023</td>
</tr>
<tr>
<td>00-70-0</td>
<td>CAL-TM-0070-0001-0009</td>
<td>Caltech</td>
<td>California</td>
<td>University of California</td>
<td>0-12</td>
<td>12-15</td>
<td>15-20</td>
<td>Journal of Chemical Society</td>
<td>00-70-0</td>
<td>12-23-2022</td>
<td>12-31-2023</td>
</tr>
<tr>
<td>00-70-0</td>
<td>CAL-TM-0070-0001-0010</td>
<td>Caltech</td>
<td>California</td>
<td>University of California</td>
<td>0-12</td>
<td>12-15</td>
<td>15-20</td>
<td>Journal of Chemical Society</td>
<td>00-70-0</td>
<td>12-23-2022</td>
<td>12-31-2023</td>
</tr>
</tbody>
</table>
Software Maintenance

- Adoption of standards
- Encapsulation of components
  - minimise coupling to reduce software complexity
  - regular monitoring of architectural dependencies
- Avoid system-dependent solutions in the source code as much as possible
  - centralise system configuration management
  - modular structure for architecture setups
- Avoid use of too “advanced” language features to maximise porting
- Traceability of updates
  - history files & regular tagging
  - disentangle development from bug-fixes
Customer Support

- Terms of the User Support are defined in the Memorandum of Understanding (MoU)
- Effort shared among WGs
  - contact persons defined for each WG
  - acting as experts in their specific domain
  - joint meetings with users and developers
- Problem Tracking System (Bugzilla) available to users
  - flexible design allowing easy customisation for Geant4
  - tokens automatically assigned to responsible persons
  - 300 reports submitted since tool in production
  - ongoing process improvement
- On-line documentation, training and FAQ on Web
- Source code and binaries available on Web and AFS
- Hypernews user forum available (hosted by SLAC)
Documentation

- Six user manuals available on-line
  - Introduction to Geant4
  - Installation Guide
  - User’s Guide for Application Developers
  - User’s Guide for Toolkit Developers
  - Physics Reference Manual
  - Software Reference Manual

- User examples: novice, extended, advanced
- Training kit: three module-structured courses
- Design documents
- Defined policy for update
Configuration Management - releases

- Defined policy for *major* and *minor* releases
  - 4 major releases, 4 minor releases, 6 patches published since in production (December ‘98)
  - policy periodically revised and updated

- Development releases distributed monthly to collaborators and developers
  - additional development releases if necessary

- Close collaboration with System Testing Team
  - acceptance tests, part also of system tests, are also run independently by the release manager

- Prompt collaboration from developers required during the public release phase
Software Process Improvement (SPI)

- Understand, determine and establish applicable procedures to Software development and maintenance of the software
- Make SPI a Software Process *life-cycle driven*

Primary life-cycle processes:
- guarantee that the code quality will not degrade with time: SPI actions associated with a regular QA activity
- assure that coupling will not increase with the growing complexity of the software

Improve overall usability and robustness of applications: improve quality, maintainability and reliability of the code

Assure continuity and integration of regular system testing within the normal Software development activity
Software Process Improvement (SPI)

- (Chosen) Domains of applicability in Geant4:
  - Q/A & Optimisation activity
    - applied to the software product in either global and component domain related context
  - Analysis & Design software cycle
    - identify the well established OOP procedure for development and maintenance – assessment based on ISO-15504
  - Testing
    - assure constant improvement and continuity to system testing

- Action for improvement identified
  - plan for SPI established
  - progressive implementation
Future evolutions

- **Make SPI part of the software life-cycle**
- Consider monitoring progress of the SPI program
  - regular check-points at *Category-Coordinator meetings*
    - regular update of status:
    - include activities addressing SPI in the Collaboration Workshops
- **Iterate new assessments in future**
  - extend assessment to uncovered (or partially covered) domains (testing, documentation, Software Management)
  - try improving Capability level
Conclusions

- Geant4, a challenging project for applying Software Processes
- Current strategy demonstrated to be effective and flexible
  - far from being perfect!
    - requires continuous monitoring and improvement
    - SPI must be life-cycle driven
  - organisational alignment