The Rules of the ISAC - Developers

The ISAC-Team
isac@ist.tugraz.at
Institute for SoftwareTechnology
University of Technology, Graz, Austria
July 11, 2007

Contents

1 The ISAC charta .................................................. 2

2 Testdriven development ....................................... 4
   2.1 Testdriven development in Java .......................... 5
   2.2 Log4j and Chainsaw ......................................... 6
   2.3 Testdriven development in SML ................................ 7

3 The coding standards .......................................... 8
   3.1 Task tags ...................................................... 8
   3.2 Name tags ...................................................... 8
   3.3 Coding standards for Java .................................. 9
   3.4 Coding standards for SML .................................. 13

4 ISAC documents ................................................ 13
   4.1 Survey on the documents .................................. 13
   4.2 Standards for the documentation .......................... 15
   4.3 Final reports ................................................ 17

5 Checklists ...................................................... 18
   5.1 Checklist for assigning a sub-task ........................ 19
   5.2 Checklist for the final hand-over of a sub-task ............ 20

This document contains all rules agreed upon by the members of the ISAC-team. The wide range of these rules will supposedly lead to a split into several documents in the future.
1 The \textit{ISAC} charta

\textit{ISAC} is dedicated to learning as an essence of human being. On the one hand learning is concerned with \textit{individual growth}, with reflection about the part of human thinking which can be mechanized by mathematics, with gaining insight into foundations of our technology oriented society — issues \textit{ISAC} wants to contribute with a novel kind of math tutoring software. On the other hand, learning is a \textit{social activity} in various ways: it concerns cooperation between the learner and some kind of teacher, between teachers and media producers, between educational administrators and course designers — issues \textit{ISAC} wants to contribute with a novel kind of math authoring software. And last not least the \textit{ISAC}-project is a learning community itself, embedded into the \textit{development in science} as a collective kind of learning.

In the future, an advisory board shall be engaged to balance these issues. Presently the \textit{ISAC}-project is primarily engaged into software development, and for this part the rules are set up first below.

1. \textit{ISAC} is an academic open-source project on learning mathematics.

2. The charta determines the administrative rules (pt.3.), the rights and obligations of the members of the (pt.9.) \textit{ISAC}-team with respect to the development process and the (pt.4.) products resulting from this process within the \textit{ISAC}-project.

3. Changes of the rules of the charta are set up to acceptance of two thirds of the (pt.10.) active members of the \textit{ISAC}-team in a (pt.12.) \textit{ISAC}-meeting. Proposals for changes have to be made public one week ahead of a decision in isac@ist.tugraz.at. In case of parity of votes the (pt.11.) \textit{ISAC}-admin decides.

4. The \textit{ISAC}-products are common property of the members of the (pt.9.) \textit{ISAC}-team, where each person holds the copyright on the code he or she is author of. The \textit{ISAC}-products comprise the \textit{ISAC} mathematics kernel, the \textit{ISAC} tutoring system, the \textit{ISAC} authoring system the \textit{ISAC} web-reader and \textit{ISAC} content.

Detailed rules for cases of re-engineering will be established in time these cases will come up.

5. \textit{ISAC} is an open source project under GNU public license; the \textit{ISAC}-project follows the idea, that educational software should be free for everyone involved in learning and teaching.

However, if \textit{ISAC}-content, probably developed outside the \textit{ISAC}-project, is commercially used, \textit{ISAC} will ensure fair sharing of profit with the \textit{ISAC}-team. A procedure for such cases will be established in time these cases will come up.

6. A certain sub-task of the \textit{ISAC}-project is defined between an aspirant and the \textit{ISAC}-admin (usually by agreement on certain JUnit tests) following the checklist 5.1. Ninety percent of this task are covered by this initial agreement; ten percent may
be required for tasks from the todo-list (pt.15.) urgently needed for accomplishing general goals of the $\text{ISAC}$-project.

By the agreement on the sub-task the aspirant becomes an active member (pt.10.) of the $\text{ISAC}$-team (pt.9.).

7. **The responsibility for parts of code** is given at the agreement on the sub-task (pt.6). The responsibility is fixed for a list of directories, where the member is allowed to create new files (and subdirectories in agreement with the (pt.11.) $\text{ISAC}$-admin) and/or a list for already existing files.

If the request for changes in the code concerns files of one other active member, then the change is due to the agreement between the two members. If the change concerns more than one colleague, the agreement is up to involve the $\text{ISAC}$-admin, usually at a (pt.13.) team-day. The $\text{ISAC}$-admin is responsible for all code not in responsibility of an active member of the $\text{ISAC}$-team.

8. **The cvs repository** contains code with all tests running (in SML as well as JUnit-tests in Java). In order to sustain this state, these steps must be followed:

   (a) Merge new code with the repository by **update**.
   (b) If there are conflicts, clarify them with the owner of the respective file (see pt.7).
   (c) Run all tests (either in SML or in Java; in case of changes in the Java-SML interface both).
   (d) If some tests do not run, contact the owner of the respective testcase (see pt.7).
   (e) If there are no more conflicts and all tests are running, **commit** the new code.
   (f) The **commit** has to be accompanied with a comment of one or two lines. If the new code is related to discussions in an $\text{ISAC}$-meeting, the comment has to reference the respective protocol (pt.14).
   (g) Renamed files and directories must be commitet with a comment containing 'oldname-¿newname' in order to make the history tracable.

When formulating the comments in commits remember: *what* has been changed is implied by the changes (and these are recorded by cvs automatically), usually it is more informative *why* something had be updated, or at which occasion some new code has bee added.

9. **The $\text{ISAC}$-team comprises** all persons who ever have obtained a task (pt.6.) from the (pt.11.) $\text{ISAC}$-admin and who have received admission from two thirds of the (pt.10.) active members in an (pt.12.) $\text{ISAC}$-meeting.

10. **The active members** of $\text{ISAC}$-team are those who have not yet left the active development process, usually on completion of their thesis or written report. The membership can be abandoned like any other rule (pt.3). An active member is discharged from this sub-task (pt.6) due to a procedure described in sect.5.2.
11. **The ISAC-admin** is the member of the ISAC-team supervising the whole development process and the adherence to the rules.

   He prosecutes breach of the rules by assigning a task, adequate to the annoyance caused in the team, from the todo-list (pt.15) to the malefactor.

   In case of absence the ISAC-admin has to determine a representative. In case of permanent unavailability the ISAC-meeting has to determine a new ISAC-admin.

12. An ISAC-meeting requires the presence of two thirds of the members of the ISAC-team, the ISAC-admin and the announcement like a rule (see above). The task of the ISAC-meeting is to change rules of this charta including the active membership.

13. A team-day obliges each active member of the ISAC-team to be present one certain day a week. This day is fixed at the beginning of a semester once for a semester.

   The team-day serves discussing interfaces and other topics of common interest, personal concerns of the members of the ISAC-team, pair programming and ISAC-meetings if required. A topic of common interest should be published in isac@ist.tugraz.at at least 3 days ahead.

14. A protocol is written for each ISAC-meeting and for points of common interest at a team-day. It serves the information of active members, who could not take part in the ISAC-meeting and for later lookup. Thus the protocol briefly describes the points of discussion and certainly contains all decisions.

   The protocol is written by the active members of the ISAC-team all around in alphabetical order. It is stored in the cvs isac/admin, thus allowing for comments.

15. A todo-list contains tasks outside the sub-tasks defined according to pt.6. Entries in the list are accepted by the ISAC-admin or by affirmation of two thirds in an ISAC-meeting.

   Tasks on this list are assigned to members of the ISAC-team by the ISAC-admin for urgent reasons w.r.t. general ISAC goals or for prosecution of breach of rules.

2 **Testdriven development**

   ISAC is a research and development projekt; as such it has to deal with open research questions and changing requirements. Together with the middle size of the ISAC-team this gives the major prerequisites for a development process following the ideas of ‘extreme programming’ [Bec00].

   In particular, all sub-tasks of development are defined by means of functional tests together with the ISAC-admin.
2.1 Test-driven development in Java

Here are the rules for the part of development involving Java.

1. The subdirectories of src/java-tests are kept an exact sub-tree of src/java. Sub-tree means, that these subdirectories in src/java-tests/* are omitted (w.r.t. src/java/*) which are not needed for holding testcases and/or testsuites.

   There are two exceptions to this rule:
   (a) src/java-tests/isac/functest see (8.) below
   (b) src/java-tests/isac/sml contains checks of the XML-output of the SML-kernel.

2. Each testcase resides in that subdirectory of src/java-tests, where the tested class resides in the respective subdirectory of src/java. (Thus, below we do not distinguish between 'subdirectories of src/java-tests/*' and 'subdirectories of src/java/*'). Thus we can name both src/java/isac/* and src/java-tests/isac/* for short ISAC/*.

3. A testsuite for class Yyy.java is named TestYyy.java, and the testcases for a method Yyy.xxx in TestYyy.java are named testXxx*. Mock-objects are named MockXxx where Xxx is the name of an existing class.

4. If a testcase refers to methods of several objects, then it resides on the common root-directory of the objects' directories. This root is src/java-tests/isac ultimately. TODO what if there will be too many testcases?

5. All testcases and testsuites in a directory ISAC/*/aaa/ are called by a specific testsuite /ISAC/*/aaa/Testall.java, and if there are subdirectories ISAC/*/aaa/?, this specific testsuite calls the testsuites ISAC/*/aaa/?/Testall.java (and does not go several levels deeper in the file hierarchy). Thus ISAC/*/aaa/Testall.java contains exactly one additional call to ISAC/*/aaa/?/Testall.java per immediate subdirectory ISAC/*/aaa/?.

6. The testsuite ISAC/Testall.java is the root of the calling hierarchy and calls all existent ISAC/*/Testall.java cascading down all tests, according to (5.), one Testall.java per level.

7. Thus the implementor of a testcase for class ISAC.*.aaa.Yyy is responsible that

   (a) the testcase is ISAC.*.aaa.TestYyy.java according to (2.) and (3.).
   (b) This amounts to either

      i. the directory src/java-tests/isac/*/aaa/ does already exist. Then the implementor has to
A. insert a call of his TestYyy.java into src.java-tests.isac.*.aaa.Testall.java (i.e. short ISAC.*.aaa.Testall.java)

ii. or directory src/java-tests/isac/*/aaa/Testall.java does not exist.

Then the implementor has to

A. create directory src/java-tests/isac/*/aaa/
B. create a testsuite ISAC.*.aaa.Testall.java calling ISAC.*.aaa.TestYyy.java
C. add a call of ISAC.*.aaa.Testall.java to ISAC.*.Testall.java, which has been brought to existence applying (5.) recursively.

8. Functional tests are kept in ISAC/functest. They relate to the use-cases in the isac-docu.tex. In order to allow cross-referencing, the whole \LaTeX-label must be used in the code, e.g. \textbackslash label\{SPECIFY:check\} (the numbering is useless due to ongoing changes to isac-docu.tex).

9. Each functional test is a separate JUnit-test. Thus it can be referenced within javadoc by @see, as can be referenced JUnit-tests themselves.

10. Interlinking of unit-tests (with the javadoc @see) is desirable.

(a) One obvious and thus mandatory way originates from the proceeding in test-driven development:
Development starts with functional tests, usually followed by JUnit-tests covering parts of the function: these JUnit-tests must be interlinked bidirectionally with the respective functional test. (i.e. a JUnit-test should point at least at one functional test.

(b) If a JUnit-test ('parent') covers several other JUnit-tests ('children'), then this relation should be documented by bidirectional links between parent and children.

11. TODO get experience, if the tests are fast enough be run parallel to module-development; then consider how to separate them appropriately.

2.2 Log4j and Chainsaw

Debuggers (in particular the one implemented in Eclipse) have troubles with ISAC’s modules running in different (virtual) machines — stepping through the code gets stuck in javaRMI.

Thus we use Log4j and Chainsaw for debugging across modules. ISAC uses Chainsaw’s priority levels as proposed at www.vipan.com/htdocs/log4jhelp.html: Log4j by default can log messages with five priority levels.

1. Use debug to write debugging messages which should not be printed when the application is in production.

2. Info: ISAC uses this mode for presenting a survey on the communication between the modules.
3. Use warn for warning messages which are logged to some log but the application is able to carry on without a problem.

4. Use error for application error messages which are also logged to some log but, still, the application can hobble along. Such as when some administrator-supplied configuration parameter is incorrect and you fall back to using some hard-coded default value.

5. Use fatal for critical messages, after logging of which the application quits abnormally.

All levels higher than debug are in the responsibility of the IS AC-admin. Thus casual overriding the usage above must be repaired by the IS AC-member on completion of his sub-task.

The logger must be declared as

```java
private static final Logger logger = Logger.getLogger(....class.getName());
```

Each call must be guarded by the if as in

```java
if (logger.isDebugEnabled())
    logger.debug(.....);
```

At level info the following abbreviations are used:

<table>
<thead>
<tr>
<th>abbreviation</th>
<th>module/class</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>WA</td>
<td>WindowApplication</td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>BrowserFrame</td>
<td></td>
</tr>
<tr>
<td>WS</td>
<td>Worksheet</td>
<td></td>
</tr>
<tr>
<td>SM</td>
<td>SessionManager</td>
<td>modules between gui and mathengine</td>
</tr>
<tr>
<td>SE</td>
<td>Session</td>
<td>? unify with SM ?</td>
</tr>
<tr>
<td>UM</td>
<td>UserManager</td>
<td>? remove ?</td>
</tr>
<tr>
<td>DG</td>
<td>DialogGuide</td>
<td>superordinate concept of BD, WD – remove ?</td>
</tr>
<tr>
<td>BD</td>
<td>BrowserDialog</td>
<td></td>
</tr>
<tr>
<td>WD</td>
<td>WorksheetDialog</td>
<td></td>
</tr>
<tr>
<td>BR</td>
<td>Bridge</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Abbreviations for survey on modules in the logger

The messages concerning SM...DG are indented 2 spaces, the BR is indented 4 spaces, WA... is not indented.

### 2.3 Test-driven development in SML

TODO
3 The coding standards

3.1 Task tags

The following task tags are used for both, for Java and for SML.

**FIX*ME** tags locations in the code where some *existing functionality* is established by a short-cut or a hack.

- **FIXME** has low priority, i.e. the fix need not made during the sub-task (see pt.6 of the ZS4C charta).
- **FIXXME** has normal priority, i.e. the fix should be made during the sub-task.
- **FIXXXME** has high priority, i.e. the fix should be made as soon as possible.

**FIXME**s need to be discussed at the final hand-over (see the checklist 5.2).

**TODO** tags locations in the code where *some functionality is missing*.

- **TODO** has low priority, e.g. it is used by eclipse’s code generator; the latter should be removed as soon as possible.
- **TOODO** has normal priority, i.e. the fix should be made during the sub-task.
- **TOOODO** has high priority, i.e. the fix should be made as soon as possible.

**TODO**s need to be discussed at the final hand-over (see the checklist 5.2).

If an author of a **FIX*ME** or a **TODO** sets the tag outside his part of responsibility to code (see pt.7 of the ZS4C charta), he has to follow the coding standards pt.5 on p.10.

3.2 Name tags

Name tags serve *short comments*, see the coding standards, eg. pt.5 on p.10 or 14 on p.11. The name tags of the members of the ZS4C-team are so far . . .
<table>
<thead>
<tr>
<th>name tag</th>
<th>username</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>AG</td>
<td>agriesma</td>
<td>Andreas Griesmayer</td>
</tr>
<tr>
<td>AK</td>
<td>akremp</td>
<td>Alan Krempler</td>
</tr>
<tr>
<td>CR</td>
<td>croppos?</td>
<td>Christian Ropposch</td>
</tr>
<tr>
<td>GK</td>
<td>gkompach</td>
<td>Georg Kompacher</td>
</tr>
<tr>
<td>GS</td>
<td>gschoet?</td>
<td>Günther Schröttner</td>
</tr>
<tr>
<td>LK</td>
<td>akirchest</td>
<td>Alois Kirchsteiger</td>
</tr>
<tr>
<td>JL</td>
<td>jloinig</td>
<td>Johannes Loinig</td>
</tr>
<tr>
<td>MG</td>
<td>mgold</td>
<td>Matthias Goldgruber</td>
</tr>
<tr>
<td>MH</td>
<td>mhochrei</td>
<td>Mario Hochreiter</td>
</tr>
<tr>
<td>MK</td>
<td>mkoschuc</td>
<td>Manuel Koschuch</td>
</tr>
<tr>
<td>ML</td>
<td>mlang</td>
<td>Martin Lang</td>
</tr>
<tr>
<td>NC</td>
<td>nsimic</td>
<td>Nebojsa Simic</td>
</tr>
<tr>
<td>RG</td>
<td>rgradisc</td>
<td>Richard Gradischnegg</td>
</tr>
<tr>
<td>RK</td>
<td>rkoenig</td>
<td>Robert Könighofer</td>
</tr>
<tr>
<td>RL</td>
<td>rlang</td>
<td>Richard Lang</td>
</tr>
<tr>
<td>SK</td>
<td></td>
<td>Stefan Karnel</td>
</tr>
<tr>
<td>TF</td>
<td>tfink</td>
<td>Thomas Fink</td>
</tr>
<tr>
<td>TO</td>
<td>tober</td>
<td>Thomas Oberhuber</td>
</tr>
<tr>
<td>WK</td>
<td>wkandl</td>
<td>Wolfgang Kandlbauer</td>
</tr>
<tr>
<td>WN</td>
<td>wneuper</td>
<td>Walther Neuper</td>
</tr>
</tbody>
</table>

The username is used by the cvs versioning system.

### 3.3 Coding standards for Java

The following closely resembles the ‘Dinopolis Java Coding Convention’.

1. The language for code is English. This applies for all names and identifiers in the code as well as for all comments.

2. Avoid the use of block comments /* ... */ in the source code and use the line comments //... instead. This makes the source code less fragile to erroneous deletions of code-lines and robust against the use of eclipses code-formatter. Eclipses code-formatter also badly handles //-comments if they are at the end of a long line; such comments should be above the respective line.

3. If it is absolutely necessary to put comments in the code to describe algorithmic details proceed the according code fragment with a comment block rather than spreading the comments across the code fragment.

4. If it is absolutely necessary to clarify non-obvious code write short comments at the end of the appropriate code line. Nevertheless whenever such a comment seems necessary think twice if there is a better obvious solution that doesn’t need a comment!
5. In exceptional cases (e.g. if the author is not an active member of the ISAC-team anymore) a comment may be added to a piece of code by someone who is not the author. In this case the comment has to be marked with NNyyymmdd, where NN follows the table in sect.3.2.

6. When describing a design pattern the name of the book which deals with this pattern should be cited (e.g. [Gamma et al. 1998]).

7. When describing algorithms the names of the book which deals with these algorithms and datastructures should be cited (e.g. [Sedgewick 1992]).

8. The source code for every class (even for non-public classes) should reside in a file of its own. The only exception to this rule are inner classes and anonymous classes as it is per definition impossible to put them in files of their own.

    /*
     * @author <author>, member of the ISAC-team,
     * Copyright (c) ${year} by <author>
     * created ${date} ${time}
     * Institute for Softwaretechnology, Graz University of Technology, Austria.
     * 
     * Use is subject to PGPL license terms.
     */

9. When writing stand-alone programs the class with the main method in it should not have anything to do with the functional part of the code. The same applies for applets: The Applet class should not have anything to do with the functional part of the code.

10. Every class should be a member of a package. Classes belonging to the default package are undesired, even for testing.

11. Java import statements should be written in the following order:

    (a) Java Core API classes.
    (b) Java Extension API classes.
    (c) Classes from third party APIs.
    (d) ISAC classes.

To increase the readability of the import part, all imports should be sorted by package names, that means imported classes belonging to the same package can be found in consecutive lines. Between the three categories mentioned above a single blank line is recommended. Using wildcards in import statements makes updating of classes hard and should therefore be avoided.
12. Classes should be preceded by a Javadoc header of the following form:\footnote{Adapt eclipse: &lt;Window&gt;&lt;Preferences&gt;&lt;Java&gt;&lt;Code Style&gt;&lt;Code Templates&gt; accordingly!}

```java
/**
 * Description of the class in HTML format, if a useful link can
 * be given in the running text do this with
 * @link fully.qualified.Class#method(fully.qualified.Param})
 * @author <authorname>
 * @version <version number>
 * @see <fully.qualified.Classname#methodName(param-classes)>
 * @deprecated <if applicable write reason here, otherwise omit
 * this line>
 */
class ExampleClass {
    ...
}
```

13. Prefix methods by a Javadoc header of the following form, if the method is not given by an interface:

```java
/**
 * Description of the method in HTML format, if a link can
 * be given in the running text do this with {@link
 * fully.qualified.Classname#methodName(fully.qualified.Paramclass})
 * @param <paramname> <paramdescription>
 * @return <return value>
 * @exception <exception> <description when it is thrown>
 * @see <fully.qualified.Classname#methodName(param-classes)>
 * @deprecated <reason if applicable, otherwise omit this line>
 */
public Object myFunction(Object test_param) throws MySpecialException {
    ...
}
```

If the method is given by an interface, the description shall not repeat the related description in the interface; use @see and refine the related description if necessary.

14. If bad hacks are absolutely unavoidable for whatever reason (e.g. absolutely have to meet a deadline, etc..) they should be tagged by a hack-start and hack-end comment of the following form, NNyymmdd according to coding standard no.5:

```java
// FIXXME.NNyymmdd hack: <description of the hack>
```
[..... the hack .....]

// END hack.NNyymmdd

For (author, date) the signature described in 5 has to be used.
To facilitate a grep, the keyword FIXXME should be written in upper-case and with at least two ‘X’s. More than two are allowed and should be used for really bad hacks - as a rule of thumb: the more ‘X’s the word FIXXME contains the worse the hack is, up to a maximum of five ‘X’s. All hacks that have found their way into the code should be removed as soon as possible!

15. Name identifiers according to the following naming conventions:

**Packages:**
- lowercase

**Interfaces:**
- I FollowedByName

**Classes:**
- AllWordsCapitalizedWithoutUnderscores

**Methods:**
- firstWordLowerCaseRestCapitalizedWithoutUnderscores

**Constants (= finals):**
- ALL_UPPER_CASE_WITH_UNDERSCORES

**Class and instance member variables:**
- all_lower_case_with_underscores_and_with_trailing_underscore_

**Auto variables (=variables used locally in methods):**
- all_lower_case_with_underscores

**Exceptions:**
- ClassNameEndsWithException

Besides these general naming rules some special naming conventions apply: All methods which change properties of classes should be named setXXX. The methods returning the value of certain properties of classes can be divided into two categories: getXXX for non-boolean properties and isXXX for boolean values respectively. Example:

```java
public void setCounter(int value) {
    ...
}
public int getCounter() {
```
16. The following code structuring conventions apply:

- Write opening curly braces at the end of the preceding code.
- Write closing curly braces around code-blocks in lines of their own.
- Indent code-blocks by two spaces. Don’t use tabs for indentations but use spaces instead. Only indent the code block, not the curly braces!
- When invoking methods the opening brace always should follow the method name without any whitespaces.
- Use the eclipse formatter each time you commit a source file.

17. java-doc is generated separately for the production-code in java and for the test-cases in java-tests. Thus do NOT reference from java to java-tests — this kind of reference is implicitly documented by the naming convention which mirrors directories, files, classnames, methodnames etc., see sect.2.1.

3.4 Coding standards for SML

The following closely resembles the standards given in [Pau91].

4. IS4C documents

IS4C as an academic project relies on the motivation, the expertise and the dedication of the members of the IS4C-team. Thus the documents are kept to an absolute minimum.

4.1 Survey on the documents

The IS4C documentation of the system has the purpose to ease entering a new sub-task. Each member of the IS4C-team is challenged to contribute to the documentation within his or her sub-task to furtherly ease the entering of follow-up sub-tasks.

---

4 Adapt eclipse: <Window><Preferences><Java><Code Style><Code Formatter> accordingly!
The ISAC-charta contains all rules agreed upon by the members of the ISAC-team; see sect.1 in this document.

The ISAC-diary gives an account on the activities going on across the different groups (development of the front-end, of the mathematics engine, of math content, etc.) in the project.

The todo-list contains tasks outside the sub-tasks defined; see the ISAC-charta pt.15. Each project sometimes needs awful things to be done ;-(( in order to succeed.

The protocols are written for each ISAC-meeting and for points of common interest at a team-day (finally phase 2 convinced the team of the necessity of this document ;-) ); see the ISAC-charta pt.14.

Add-ons to the protocols may contain more voluminous comments on discussions, details of design considerations etc. than acceptable in the protocol. This add-on must have the same date in the filename as the protocol it is added on.

The work-plans are set up separately for each sub-task. There was no need for a formalized work-plan or a project plan over several sub-tasks so far.

The final reports conclude each sub-task, beeing it a seminar/project or practical part of some kind of thesis; see sect.4.3. They are source of major updates of the ISAC documentation (preferably by copy and past), and thus follow the same standards, see sect.4.2

The work-reports contain information important for continuing work related to a specific sub-task, if such information cannot becovered by TO*DOs and FIX*MEs in the code.

Administrative details on the documents: Most of the documents require versioning, thus they are located in the cvs-repository at /isac/admin and sub-directories included in the field ’name’ of the table below.
A 'member' of the IS AC-team is abbreviated by 'mem' above. Templates are held in the respective directory with the name template.* or given by the initial entry in the respective document.

### 4.2 Standards for the documentation

These standards hold for the ISAC design documents, i.e. the user requirements and the software requirements document, the architectural design and software design document, the use cases and test cases, and the final reports, see see sect. 4.3 below.

These documents are written in **\LaTeX**, which is unfamiliar with many authors; thus the standard is kept to a minimum of sophistication. The aim is to provide for easy merging (parts of) the final reports into the ISAC-docu [iT02].

The structure into parts, chapters, sections is given by the ISAC-documentation. The structure of the final reports should take the same levels.

Definitions are already given in the file isac-docu.tex (the definitions will be extracted into a separated file preamble.tex as soon as some details with separate compilation are solved, see '\LaTeXXing' on p.17 below). In order to avoid conflicts, they must all be copied into the separate reports at the very beginning of writing!

Logos i.e. ISAC and ISAC are fairly primitive \LaTeX-constructs. Both require a { } for separating the subsequent word; ISAC (fixed size !) is for use within paragraphs, ISAC
for headlines.

**User-requirements, software-requirements and use-cases** have all their respective definitions by `\newcommand` and `\newtheorem` in `preamble.tex`; they *must* be used. How to use, just look into the `isac-docu.tex` files! See also 'labels' below. ⁵

**Labels and files-names** must be headed by the name tag of the author, see sect.3.2 on p.8 — this is by no means an elegant way of avoiding conflicts when integrating the final reports into the `isac-docu.tex` files, but who knows a better one?

Moreover, labels of user-requirements start with `UR`, software-requirements start with `UR` and use-cases start with `UC`. Thus a typical label is `\label{UR.WN-short-description}` and the respective reference `\ref{UR.WN-short-description}` — note the preceding `UR`.

Files-names *must not* contain underscores (`,`) for the (rare) cases they have to be cited within LATEX (otherwise we would have to use math-mode).

**Figures** should be generated using `xfig`. The source files should have the same name as the `*.eps`-files generated for LATEX, and they both must be located in a sub-directory `fig` of the root-directory of the respective report (as is with `isac-docu.tex`); And these file names must start with a name tag and must not contain underscores according to 'labels and files-names' above.

**Diagrams** should be created with the tools from [http://uml.sourceforge.net/index.php](http://uml.sourceforge.net/index.php). Umbrello is open-source and seems to become *the* up-coming tool for UML-modeling.

**Reader’s marks** are used in the (rare) cases, where a member of the ISAC-team is authorized by the ISAC-admin to edit parts of the ISAC-documents directly. Then the ISAC-admin will use the following, well proven, reader’s marks:

```
% legend to the reader’s marks:
%
% [] the brackets enclose comments additional to,
%    and not belonging to the text
%
% {} the braces enclose exact proposals for new text,
%    which are embedded into comments.
%
% / marks a character to be deleted in the line _above_
%
% ^ points to a certain position in the line above,
% usually concerning a comment or an insertion
```

⁵[Kre05] proposed more elegant definitions for them, which shall be introduced in the future.
The same marks are used for comments of the LSAC-admin within the final reports, if desired.

\textbf{\LaTeX \textxing} of the LSAC-documentation is done with 'latex isac-docu' and 'latex math-eng'. And each of the documents must be compiled with the \LaTeX\-system actually installed at IST — this is an indispensible prerequisite for maintainance of the documentation.

Each part of the LSAC-documentation can be \LaTeX\-xed separately, the User Requirements Document by 'latex urd' etc. This is due to a mechanism based on the file common.tex copied from [Dil93].

\textbf{\Bib\textxing} of the LSAC-documentation is done with 'bibtex isac-docu' and 'bibtex math-eng'. The related bib-files are the files isac/doc/bib/isac.bib and isac/doc/bib/from-theses maintained by the LSAC-admin.

All LSAC-related publications are in isac/doc/bib/isac.bib (otherwise urge the LSAC-admin !), and \texttt{\cite{xxx}} must use the labels xxx already given.

Bib-tex files must be located in a sub-directory bib of the root-directory of the respective report (as is with isac-docu.tex).

\section{4.3 Final reports}

Most of the members of the LSAC-team work on sub-projects in LSAC within their regular studies, comprising a ‘Diplomarbeit’ (diploma thesis), a ‘Software-Projekt und Bakk.-Arbeit B’, a ‘Seminar/Projekt’, or a ‘Praxis-Semester’. The latter usually is continued into a diploma thesis, too. Thus most of the sub-tasks end up with a final written report.

In the sequel there are supporting aids and rules for these reports and theses.

\textbf{Support for writing} the final reports is given in several ways, most of them contained in the versioning system, the CVS as checked out into isac/. There are

- a lot of documents and papers is avaialble on LSACs webspace \url{www.ist.tugraz.at/projects/isac/} via download. Members of the LSAC-team obtain these papers, including \LaTeX\-sources, figures etc. from the CVS at isac/doc or directly from the LSAC-admin

- in particular, surveys on LSAC, introductions to LSAC, proposals on how to locate sub-projects within LSAC etc. directly from the LSAC-admin

- stylesheets for the reports, including the definitions of LSAC, LSAC etc. in the CVS at isac/doc.

- a bib-file in the CVS at isac/doc/bib for easy generation of bibliographies. This file is maintained by the LSAC-admin. Further bib-files can be supplied by the LSAC-admin.
Coordination with the ISAC-documentation is both, helpful for writing a report or thesis, and mandatory in order to keep the documentation up to date. The following rules guide the coordination.

1. *Terms used in the ISAC-project*, as contained in an appendix of the ISAC-documentation, provide for efficiency in internal communication and for uniformity and tracability in presentation to the outside world. Thus, in particular, these terms *must* be used in reports and in theses.

2. Writing access to the ISAC-documentation in the cvs at *isac/doc* is exclusively with the ISAC-admin (who may delegate certain tasks).

3. The *Requirements Document*, both the user requirements and the software requirements in the ISAC-documentation, must provide justification for all design decisions in a report. If gaps in the requirements document become apparent, they have to be filled in coordination with the ISAC-admin.

4. The *Architectural Design Document* may overlap with design considerations in a report or thesis. Respective parts of the document may be copied into the report (and cited). And if there are changes or refinements in the design, the respective parts of the report will be copied into the document in cooperation with the ISAC-admin.

5. The *Software Design Document* usually will be refined, updated and completed by parts of a report; again, these parts of the report will be copied into the document in cooperation with the ISAC-admin.

6. The *Usecases* are shifted into the code as soon as they are implemented: the practical parts of the project/seminar or the thesis are defined by functional tests according to sect.2. Nevertheless, a usecase must be referenced by the full \LaTeX-label, e.g. `\label{SPECIFY:check}`.

**TODO.WN050527 after decision for/against doxygen:**

7. If a major part is copied from a final report to the ISAC-documentation, then such a part is marked within both, in the source (the report) and in the destination (the documentation), e.g.

```plaintext
\%WN050518---AK04:thesis.p.67-76->isac-SDD------BEGIN don’t remove line
\%WN050518---AK04:thesis.p.67-76->isac-SDD------END don’t remove line
```

5 Checklists

The first two checklists concern the begin and the end of a sub-task.
5.1 Checklist for assigning a sub-task

This checklist concerns the adoption of a sub-task as defined in pt.6 on p.2.

1. the ISAC-admin presents the essence and the most important rules of the ISAC-charta

2. agree on an individual access to the practice of the rules in the ISAC-charta

3. receive an account at the Institute for Softwaretechnology, IST (the ISAC/admin arranges an appointment with the IST-admin).

4. describe the sub-task: this usually is an interactive procedure within the team lasting for some time. Anyway, it shall be finalised within 3 weeks. The description depends on the specific sub-task and partially on the working style of the current ISAC-team. Usually the description comprises some of the following tasks:

5. relate the sub-task to existing use-cases

6. relate the sub-task to existing parts of the isac-docsu.tex, i.e. the actual version of www.ist.tugraz.at/projects/isac/publ/isac-docu.ps.gz

7. assign the source-directories and files the new member is responsible for

8. fix an appointment for a work-plan describing appropriate milestones for the sub-task (should be within 4 weeks in general)

9. fix the type of documentation; this may be a thesis (see sect.4.3) or a work-report only (see sect.4.1). The former is expected to be discussed with the ISAC-admin in order to ensure compatibility with the ISAC-documents; see sect.4.2 for standards.

10. mail a personal web-page for www.ist.tugraz.at/projects/isac/www/content/team.html (should be within 4 weeks in general); the ISAC-admin copies it into the webspace.

11. the ISAC-admin assigns a name tag to the new member (see the table in sect.3.2)

12. check if the administrative duties with the university are accomplished (enrolment for semin/project, for Bakk or Diploma thesis etc.

13. introduce the new member of the ISAC-team on an ISAC-meeting (pt.12 on p.4)

14. introduce the new member to other members of the ISAC-team responsible for related sub-tasks.

15. hand-over the work-plan to the ISAC-admin (who approves and publishes it – see sect.4.1)

16. (ISAC-admin: update the web-pages, publish work-plan, announce in the isac-diary.tex)
5.2 Checklist for the final hand-over of a sub-task

This checklist concerns the final hand-over of a sub-task usually releasing an active member (pt.10 on p.3).

1. check your code, i.e. the code you are responsible for (pt.7 of the IS4C-charta)

   (a) general checks
   i. the coding standards are met
   ii. no warnings / errors
   iii. remove all obsolete FIX*MEs and TO*DO (and oly these !) and comment the others
   iv. all outcommented code must have an extra comment indicating the reason.

   (b) java-specific checks
   i. the coding standards are met, in particular, all important classes and methods have a java-doc comment
   ii. java-doc compiles both, the java-directory and the java-tests-directory
      (separated following pt.17 on p.13)
   iii. check your code via javadoc: each class and each method with a brief and relevant comment
   iv. remove all calls of the logger, except those with status 'fatal' which have (exclusively !) been managed by the IS4C-admin.
   v. reduce the number of System.out.println's to a reasonable amount of output, which might help (and not overwhelm) your successors in development.

   (c) sml-specific checks
   i. TODO

2. check the usecases done or not done (and comment the latter in the (JUnit-) testcase !)

3. finalize the documentation, i.e. the final report and/or the assigned parts of the isac-docu:

   (a) check w.r.t. the standards in sect.4.2

   (b) actually latex the report on the IST-system (indispensable for reuse of the text and the figures !) and generate a ps-file by dvips da-NN.dvi -o, as an option a pdf-file additionally (which causes additional effort with the figures, in general).

   (c) hand over the sources to the IS4C-admin for publication and/or for integration into the IS4C-docu.

4. finish your work-plan to your sub-task, see sect.4.1.
5. make an appointment with the ISAC-admin and discuss ...

   (a) the work-plan
   (b) specific points in your code
   (c) each $\text{FIX*ME}$ within your code
   (d) each $\text{TO*DO}$ in this code
   (e) the test-cases
   (f) the final-report

This meeting may take more than one appointment.

6. discuss the most important points from above at a team-day (pt.13 on p.4) or at an isac-meeting (pt.12 on p.4).

7. optionally deposit a final work-report (see sect.4.1) on the sub-task, on experiences with the ISAC-team etc.

8. if you are also a member of TU Graz, ask the ISAC-admin to arrange an appointment
    for giving a presentation at an IST-meeting.

9. return the keys, books, cables etc. you have from the IST.

10. say goodbye at a team-day or at an ISAC-meeting.

References


[iT02] ISAC Team. *ISAC* – user requirements document, software requirements document, architectural design document, software design document, use cases, test cases. Technical report, Institute for Softwaretechnology, University of Technology, Graz, Austria, March 2002.
