The *open*EHR Change Management Plan

Editor:{T Beale}[1]

Revision: 1.0

Pages: 37

1. Ocean Informatics Australia

© 2003-2006 The *open*EHR Foundation

The *open*EHR foundation

is an independent, non-profit community, facilitating the creation and sharing of health records by consumers and clinicians via open-source, standards-based implementations.

Founding Chairman	David Ingram, Professor of Health Informatics, CHIME, University College London
Founding Members	Dr P Schloeffel, Dr S Heard, Dr D Kalra, D Lloyd, T Beale

email: info@openEHR.org **web**: www.openEHR.org

Copyright Notice

© Copyright *open*EHR Foundation 2001 - 2005

All Rights Reserved

1. This document is protected by copyright and/or database right throughout the world and is owned by the *open*EHR Foundation.
2. You may read and print the document for private, non-commercial use.
3. You may use this document (in whole or in part) for the purposes of making presentations and education, so long as such purposes are non-commercial and are designed to comment on, further the goals of, or inform third parties about, *open*EHR.
4. You must not alter, modify, add to or delete anything from the document you use (except as is permitted in paragraphs 2 and 3 above).
5. You shall, in any use of this document, include an acknowledgement in the form:

"© Copyright *open*EHR Foundation 2001-2005. All rights reserved. www.openEHR.org"

6. This document is being provided as a service to the academic community and on a non-commercial basis. Accordingly, to the fullest extent permitted under applicable law, the *open*EHR Foundation accepts no liability and offers no warranties in relation to the materials and documentation and their content.
7. If you wish to commercialise, license, sell, distribute, use or otherwise copy the materials and documents on this site other than as provided for in paragraphs 1 to 6 above, you must comply with the terms and conditions of the *open*EHR Free Commercial Use Licence, or enter into a separate written agreement with *open*EHR Foundation covering such activities. The terms and conditions of the *open*EHR Free Commercial Use Licence can be found at http://www.openehr.org/free_commercial_use.htm

© 2003-2006 The openEHR Foundation
email: info@openEHR.org web: www.openEHR.org

Amendment Record

Issue	Details	Who	Completed
RELEASE 1.0			
1.0	Update release details.	T Beale	28 Jan 2006
0.9	Changes due to migration to subversion.	T Beale	28 Jul 2005
0.8	Split off *open*EHR Overview document.	T Beale	04 Feb 2005
0.7	Addition of sections on governance, IP and project overview.	T Beale M Darlison	10 Jan 2005
0.6	Review by UCL team	D Kalra, T Austin, N Lea, D Lloyd T Beale	10 Dec 2004
0.5.2	Review by prof. David Ingram (UCL).	D Ingram	20 Feb 2004
0.5.1	Further review by Tim Cook <Tim@openparadigms.com>.	T Beale	10 Dec 2003
0.5	Further development of ARB process.	T Beale	06 Dec 2003
0.4	Review by Tim Cook <Tim@openparadigms.com>; further development.	T Cook, T Beale	22 Nov 2003
0.3	Updated in preparation for public *open*EHR CM site launch.	T Beale	15 Nov 2003
0.2	Updated May visit UCL	T Beale	13 May 2003
0.1	Initial Writing	T Beale	20 Jan 2003

Acknowledgements

This paper has been funded by The University College, London and Ocean Informatics, Australia.

© 2003-2006 The openEHR Foundation
email: info@openEHR.org web: www.openEHR.org

© 2003-2006 The openEHR Foundation
email: info@openEHR.org web: www.openEHR.org

Table of Contents

1 Introduction ... 7
1.1 Purpose ... 7
1.2 Audience ... 7
1.3 Status ... 7
1.4 Terms and Acronyms ... 7
2 Overview ... 8
2.1 Activities ... 8
2.2 Management ... 8
3 openEHR Technical Projects ... 12
3.1 The Specification project ... 12
3.2 Reference Implementation Projects ... 14
3.3 Ad hoc Implementation Projects ... 14
4 Repository Organisation ... 16
4.1 Repository Naming ... 16
4.2 Repository Design ... 16
5 Release Management ... 18
5.1 Overview ... 18
5.2 Release Structure and Branching ... 18
5.3 Release Naming ... 20
6 Change Management ... 21
6.1 Overview ... 21
6.2 The Change Process ... 22
6.2.1 Overview ... 22
6.2.2 Problem Reporting ... 23
6.2.3 Change Request Process for openEHR Reference Projects ... 23
6.2.3.1 Workflow ... 23
6.2.3.2 CR Lifecycle ... 24
6.2.3.3 Project Group-managed CRs ... 25
6.2.3.4 Review Board-managed CRs ... 25
6.2.4 Change Request Process for ad hoc Projects ... 26
6.2.4.1 Workflow ... 26
6.2.4.2 CR Lifecycle ... 27
6.2.4.3 CR Management ... 27
7 Tools ... 29
7.1 Overview ... 29
7.2 Configuration Management System ... 29
7.3 PR / CR Database ... 30
7.4 Publishing/Distribution ... 30
8 CI Identification ... 31
Appendix AForms ... 33
A.1 Problem Report Form ... 33
A.2 PR Form Fields ... 33
A.3 Change Request Form ... 34
A.4 CR Form Fields ... 34

© 2003-2006 The openEHR Foundation
email: info@openEHR.org web: www.openEHR.org

© 2003-2006 The openEHR Foundation
email: info@openEHR.org web: www.openEHR.org

1 Introduction

1.1 Purpose

The purpose of this document is to describe the management of *open*EHR technical projects, i.e. projects in the *open*EHR technical space (as described in the document Introducing openEHR) and carried out within the *open*EHR development environment. These projects have "controlled" deliverables, and a clear problem reporting and change request strategy, defined by this document. Two change management strategies are described: one with a "review board" for *reference* projects, and one without, for *ad hoc* projects.

1.2 Audience

The primary audience for this document is developers of specifications and software for the *open*EHR Foundation.

1.3 Status

This document is available at http://svn.openehr.org/specification/TAGS/Release-1.0/publishing/CM/CM_plan.pdf.

The latest version of this document can be found at http://svn.openehr.org/specification/TRUNK/publishing/CM/CM_plan.pdf.

1.4 Terms and Acronyms

ARB Architectural Review Board

CI Configuration Item - any controlled artifact, such as a document, source file, test case etc.

CM Configuration management

IP Intellectual property

PG Project Group

© 2003-2006 The openEHR Foundation
email: info@openEHR.org web: www.openEHR.org

2 Overview

2.1 Activities

The figure below illustrates the technical activity areas of *open*EHR, including specification and implementation projects, and delivery/deployment activities.

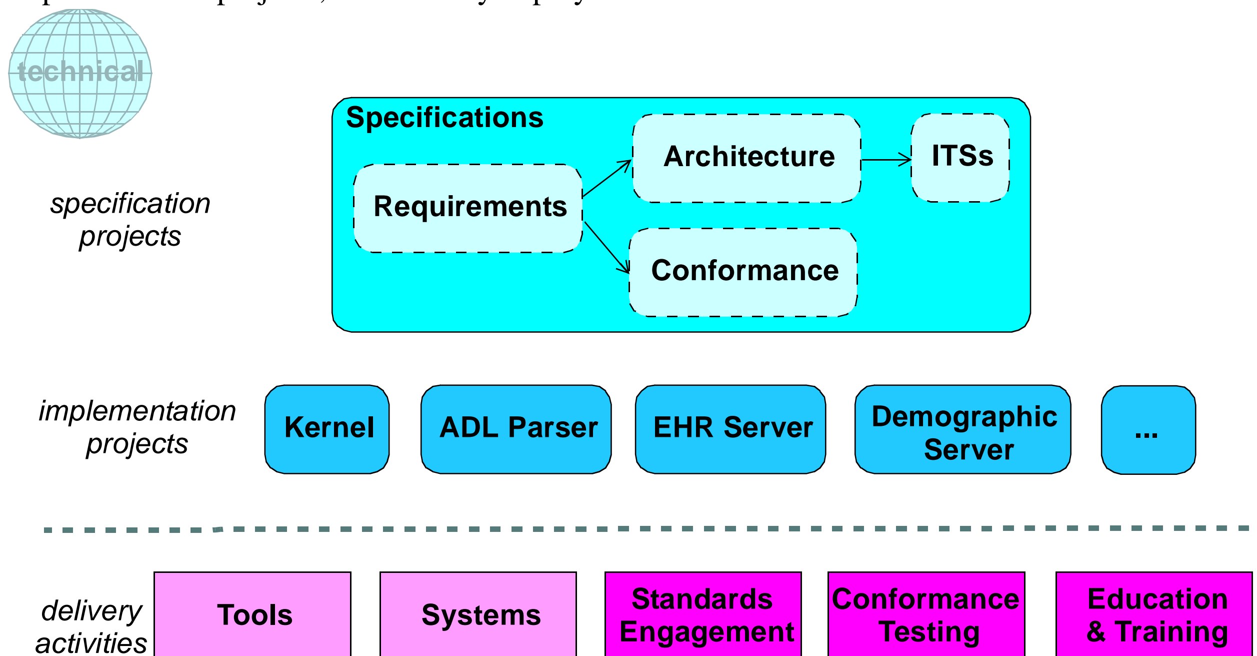

FIGURE 1 *open*EHR Technical Activities

Each solid-line bubble on the left part of the diagram is a *project* in *open*EHR. As shown, there are four major areas of activity: specification, implementation, knowledge, and delivery. The first two correspond to what most people would think of as software development; their change management is the subject of this document. The projects in these two groups are described in more detail in section 3 on page 12.

2.2 Management

In the technical space of *open*EHR, work is performed by project groups (PGs), which are in some cases overseen by the Architectural Review Board (ARB). The ARB consists of a eight or more international members of *open*EHR, all with long-term experience in an area of health informatics. The current makeup of the ARB may be found on the *open*EHR website ARB page. The ARB's function is to review and make decisions on requests for change that either have significant impact on a project, or that cannot be resolved by the project development group on its own. It operates using simple majority voting.

Project types

Project groups are groups of developers responsible for the work done on a project. There are two kinds of *open*EHR technical project - *reference* projects, and *ad hoc* projects. Reference projects develop *reference deliverables*, which constitute the "official" basis for the community to develop and test the conformance of products. All reference projects are change managed by their project group and the ARB.

© 2003-2006 The openEHR Foundation
email: info@openEHR.org web: www.openEHR.org

*Ad hoc open*EHR projects on the other hand do not generate reference specifications or implementations, and can be change-managed without recourse to the ARB. *Ad hoc* project groups will most likely be self-selecting. Anyone can become a member of a project by making themselves known to the existing group, and being given modification rights on the relevant repository. A management view of *open*EHR's technical space is illustrated in FIGURE 2 below. In this figure, reference project groups are shown connected to a board of review, while *ad hoc* projects are not.

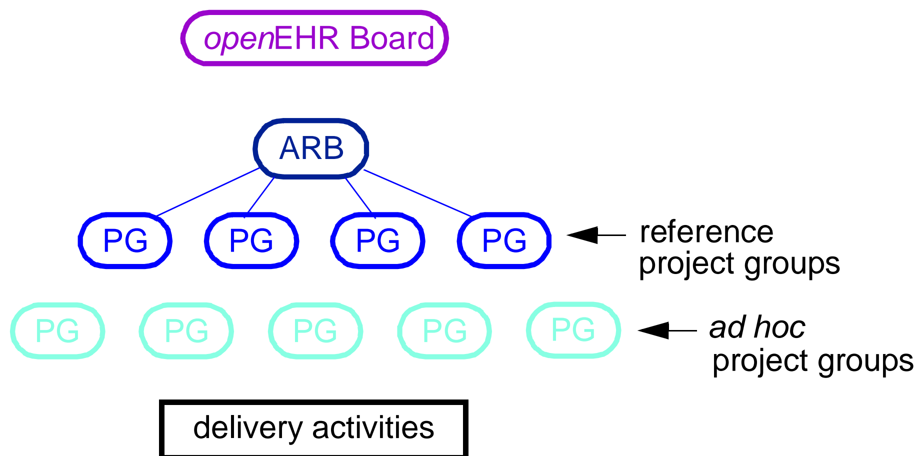

FIGURE 2 Management View of the *open*EHR Technical Space

*open*EHR Reference Deliverables

Some deliverables created in the *open*EHR technical space are "reference deliverables". These artifacts are the definitive instance in their category: the *open*EHR information and service model specifications, the implementation technology specifications (ITSs) such as XML-schemas, programming language interfaces, *open*EHR terminology, conformance test cases and *open*EHR reference implementations (e.g. parsers). Reference implementations are created either for the purposes of conformance testing, or in cases where absolutely dependable, re-usable, standard components are required. All *open*EHR reference deliverables are created by *open*EHR reference projects.

The relationship among various kinds of projects and *open*EHR/non-*open*EHR products is shown in FIGURE 3 below.

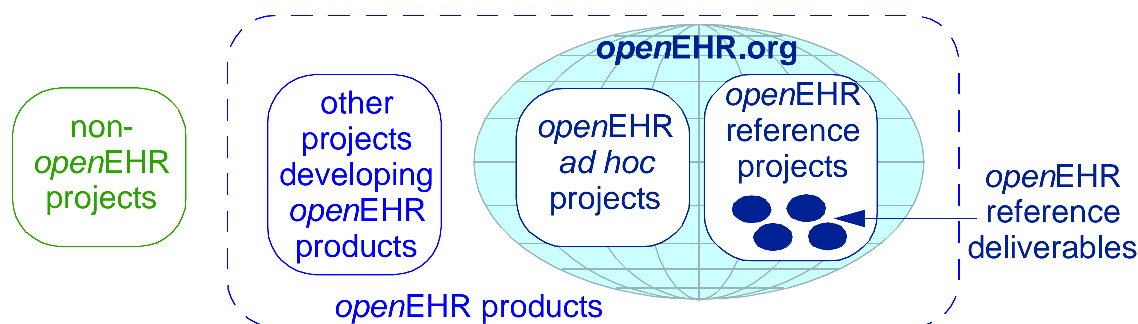

FIGURE 3 Relationship between Projects and Products

Definition of an *open*EHR Technical Project

For the purposes of managing work done under the *open*EHR banner, the notion of an "*open*EHR technical project" is explicitly defined as any project that follows this change management plan.

© 2003-2006 The openEHR Foundation
email: info@openEHR.org web: www.openEHR.org

is a) based on *open*EHR in some formal way (typically aims to build something that satisfies *open*EHR conformance criteria) and b) that agrees to do its work within the development framework offered at *open*EHR.org, defined below. All projects that develop reference deliverables are *open*EHR projects. The *open*EHR project development environment is defined by the following.

Change Management

- A standard version and change management toolset/environment. *open*EHR currently uses the open source tool `Subversion` for this purpose.
- A basic change management rule - only a member of a project team can a) create a change request, and b) make any change to the project repository. This simply means that the team always knows who is in it, and has agreed among themselves that they can make modifications. Non-team members proposing sensible modifications are likely to be asked to join the team.
- A standard Problem Report (PR) lifecycle.
- A standard Change Request (CR) lifecycle.
- A standard online tool/environment for creating and accessing CRs and PRs. CRs need to be able to be created and viewed in a standard way by developers, whilst PRs need to be created and viewed in a known place and in a sensible way by users. PRs are the public problem-logging and reporting interface for users.
- For reference projects, development is overseen by the ARB, according to the change management process described in Change Request Process for openEHR Reference Projects on page 23. For *ad hoc* projects, development may adopt the simpler non-ARB process described in Change Request Process for ad hoc Projects on page 26.

Build and Release

- The top level directory structure of implementation projects is fairly similar if not the same.
- An approach to build management that is as far as possible homogeneous across projects (facilitates developers working on more than one project). This does not necessarily have to be a single tool, but if e.g. **ant** and **make** are used, they should be used in the same way across projects.
- A standard way of distributing the software to users, particularly binaries (i.e. make it easy for non-IT users).

Intellectual Property Rights

- Copyright may optionally be transferred to the *open*EHR Foundation, converted to joint copyright with the *open*EHR Foundation, or may remain with the originating organisation or author.
- Documents (e.g. manuals) use the standard *open*EHR document licence.
- Source code uses the standard *open*EHR open source licence. This is currently the Mozilla tri-license, which is really just a meta-license allowing the user to nominate GPL, LGPL, or MPL as the licence they use the software under.
- Respect the structure of the **org.openehr** namespace by the ARB, for source code, schemas, terminologies and any other reference deliverable. Non-reference projects may not use the **org.openehr** namespace.
- Irrevocability: organisations cannot retrospectively revoke the right of the *open*EHR Foundation and community to continue to use software or other artifacts which they have developed within the *open*EHR environment (since this would contravene the terms of the

© 2003-2006 The openEHR Foundation
email: info@openEHR.org web: www.openEHR.org

license). They may of course use any such developed works as a basis for other developments. This condition ensures that neither the community (which may have come to rely on a component) nor the original developing organisation (which may have spent significant time and money on the development) lose access to the work; if the interests cease to coincide, the development is simply “forked”, and only one line remains with *open*EHR.

Projects that agree to these items will be able to take advantage of the facilities provided by the *open*EHR Foundation, including version management, build servers and a distribution server. It particularly enables smaller projects to proceed where otherwise they might not have sufficient technical resources.

This approach to development is offered as a service by *open*EHR, and of course is not a requirement of developing *open*EHR-compliant products. Development organisations are encouraged to develop *open*EHR-based products in any way they see fit. Many projects will be done by companies, universities and so on, according to their own processes, including completely commercial closed source projects.

© 2003-2006 The openEHR Foundation
email: info@openEHR.org web: www.openEHR.org

3 *open*EHR Technical Projects

3.1 The Specification project

The *open*EHR specification project includes deliverables that are considered technical specifications - i.e. that can be used either to develop further specifications, or to build systems, test plans, or other usable artifacts. The specification project includes requirements, abstract architecture, ITSs and conformance specifications as shown in the following table.

Deliverable	Component	Description
Requirements	**Requirements Base**	The Requirements Base is a repository of requirements underpinning the EHR and related functionality in the health information environment. This repository is the definitive requirements basis of *open*EHR, and will continue to evolve in time. It consists of both functional requirements and use cases.
	Conformance Statement	Conformance statement of *open*EHR with respect to existing/emerging standards, e.g. ISO TS 18308.
Architecture	**Design Principles**	Design principles of health information systems and in particular the EHR
	Reference Model (RM)	The primary set of abstract, formal specifications of *open*EHR models in the information viewpoint. These abstract expressions are independent of implementation technologies. Includes abstract Information Models for: • EHR_extract • Common • Data Structures • Data Types • Support (low level primitives)
	Service Model (SM)	The primary set of abstract, formal specifications of *open*EHR models in the computational viewpoint. These abstract expressions are independent of implementation technologies. Includes abstract Service Models for: • EHR • Demographics • Workflow • Archetype repository
	Archetype Model (AM)	Various formal specifications defining the *open*EHR archetype semantics. • Archetype Principles • Archetype Definition Language (ADL) specification • Template Definition Language (TDL) specification • Archetype Query Language (AQL) specification • Archetype Object Model (AOM)
	Archetype System	• The *open*EHR Archetype System

© 2003-2006 The openEHR Foundation
email: info@openEHR.org web: www.openEHR.org

Deliverable	Component	Description
Implementation Technology Specifications (ITSs)	**ITS-java**	• Java expression of architecture specifications
	ITS-xml_schema	• XML-schema expression of architecture specifications
	ITS-idl	• OMG IDL expression of architecture specifications
	ITS-csharp	• C# expression of architecture specifications
	etc	• other languages,
Conformance	**Conformance**	Test cases and plans for testing of conformance of *open*EHR implementations against Requirements and ITSs.

The primary models are used as the source for the published documentary form of the specifications, generally in Adobe PDF format. There is not considered to be any semantic difference between tool-based abstract model expressions and their documentary counterparts, i.e. there is no "mapping" or "conversion". The primary models are also losslessly translated to a UML-2.0 compliant XML instance form, from which all other views are generated.

Implementation Technology Specification (ITS) Components

ITSs are the concrete expressions of abstract specifications in specific implementation-oriented technologies, and are the artifacts used directly for building software and databases. They are generated via a *mapping* process, and *may have reduced semantic content*, e.g. they might not include certain abstract semantics such as functions, invariants. For example, the XML-schema ITS does not contain functions, since XML-schema is a data-oriented formalism, and does not have a way (or need) to express functions. Otherwise however, ITSs do include full coverage of all the relevant *open*EHR specifications. The two categories of ITS are as follows.

Interoperability specifications include any expression of an abstract specification in a concrete interoperability technology, including:

- IDL expressions, e.g. in OMG IDL syntax, DCE syntax Microsoft, WSDL, or other publicly available interface formalisms
- XML schema or other XML-based formalism

Implementation specifications include any expression of an abstract specification in a concrete implementation technology, including:

- any programming language. Such expressions may be code interfaces, example working code, or other code guidelines.
- any database schema language. Such expressions may include full schemas for particular database products and generic schemas for a class of product.

Change Management

In the above table, the items in the "Component" column are the items against which problem reports (PRs) are made. In general, PRs will only be raised against executable components or computable components like XML-schemas.

The abstract architecture deliverable (second major row of table) is the main driver of major releases of the specification project. That is to say, if the abstract models change in any significant way, a new release is declared. The Architectural Review Board (ARB) decides on new releases.

© 2003-2006 The openEHR Foundation
email: info@openEHR.org web: www.openEHR.org

Changes in the abstract architecture models will immediately cause changes in the ITSs, since these are the directly usable expression of the abstract architecture. However it might be some months before various implementation projects are changed to reflect specification changes.

3.2 Reference Implementation Projects

Reference implementation projects produce artifacts of which only one official version is needed. These are usually used as the basis for conformance testing, or in some cases, are core software components that must have guaranteed correctness and reliability. The following table shows a number of *open*EHR reference implementation projects.

Project	Component	Description
Tools	(various)	This project develops various tools • any converter that creates a derived artifact from an abstract one • tools for validating specifications in the reference model • tools for working with test data or test cases • archetype and template validation tools
Kernel	**Kernel in lang X**	Open source implementation of the *open*EHR reference and archetype models, as an archetype- and template-enabled data processing component.
	Wrappers in other languages	Open source implementations of wrappers of the kernel for other development languages
ADL Parser	**Eiffel ADL Parser**	Original ADL reference parser implementing current version of the ADL specification.
	Java-wrapped ADL parser	Java wrapping for the parser, implemented using JNI
	dotNet ADL parser	Dotnet edition of the parser, for MS Windows.
etc		

3.3 *Ad hoc* Implementation Projects

The *ad hoc* implementation projects produce non-reference tools or systems based on any reference deliverable, whether directly from specifications, or on existing components. Over time, there may well be multiple projects each implementing the same category of deliverable, such as an archetype editor or EHR server; products developed in this way will usually perform the same general function, but may have significantly different performance characteristics, user interface design approaches, or differ in some other way relevant to end use. The following table shows a list of possible *open*EHR projects, and their components.

Project	Component	Description
Archetype Editor		Tool for creating and viewing archetypes.
EHR server		EHR repository providing versioned Contribution interface, with transaction management
Demographic server		Demographic data repository, providing versioned Contribution interface, with transaction management

© 2003-2006 The openEHR Foundation
email: info@openEHR.org web: www.openEHR.org

Project	Component	Description
etc		

© 2003-2006 The openEHR Foundation
email: info@openEHR.org web: www.openEHR.org

4 Repository Organisation

4.1 Repository Naming

Each *open*EHR project is controlled in a separate versioned repository. Currently, the Subversion tool is used to manage repositories (see http://subversion.tigris.org and the relevant openEHR website pages). Due to the way Subversion works, each project is a separate repository (explained in detail in next section). Further, since most implementation efforts are oriented to one or other of the major technologies avalailable, repositories are also distinguished on this basis. Technologies include:

- Java and related frameworks, such as Eclipse, NetBeans, EJBx, J2EE, Hibernate etc;
- Microsoft .Net technologies, including C#.Net and VB.Net languages;
- Python and related tools such as Zope and Plone;
- The Eiffel language and technologies.

Repositories are separated on the basis of coherent projects rather than fine-grained choices of e.g. persistence framework for Java. Thus, the "Java reference kernel" project might develop one set of base libraries, and two alternative persistence mechanisms. If all of this work forms a coherent whole, it will be defined as a project, and have its own Subversion repository. On the other hand, not all things done in a given technology need be in the same repository. Knowledge tools such as archetype editors and repository tools built in Java would have their own repository, since the project is independent of the EHR kernel and server work, and developed by a different group. There is no problem for sharing libraries from one project with another - this is simply done by setting up the user workspace on the client machine appropriately, so that build scripts can find all the software.

In general the repositories are also desgined as "source-only" repositories; there is a separate repository for distributable binaries.

4.2 Repository Design

The *open*EHR subversion repositories include the following:

- **specification**: the specification project
- **knowledge**: clinical domain content, including terminology, archetypes, templates etc
- **knowledge_tools_java**: knowledge management tools built in Java, including a Java archetype editor
- **knowledge_tools_dotnet**: knowledge management tools built in .Net languages, including the Ocean Informatics VB.Net archetype editor
- **ref_impl_java**: Java implementation of *open*EHR reference and archetype models, and archetype-processing kernel
- **ref_impl_dotnet**: Microsoft .Net implementation of *open*EHR reference and archetype models, and archetype-processing kernel
- **ref_impl_eiffel**: reference implementation of *open*EHR in the Eiffel language
- **oe_distrib**: repository containing binaries and packages for distribution.

Other repositories will be defined as the need arises. The full list of projects can always be found on the "projects" button on the openEHR home page.

The internal structure of repositories tries to achieve two (sometimes contradictory) aims:

© 2003-2006 The openEHR Foundation
email: info@openEHR.org web: www.openEHR.org

- enable basic functioning of Subversoin;
- be consistent across repositories, in order to facilitate comprehension by new project members;
- be consistent with accepted conventions for the relevant implementation technology, particularly as required by build tools or other framework components.

In support of the first aim, the following top-level structure is usually used:

- **TRUNK**: mainline current development;
- **BRANCHES**: branch developments for new or alternative or test work;
- **TAGS**: named baselines (no development allowed)
- **RELEASES**: release branches, i.e. branches corresponding to stable major points in the mainline (trunk) development, whose only further allowed changes are bugfixes.

This corresponds more or less to the recommended way in Subversion of separating mainline development; the only difference is that in *open*EHR upper-case directory names have been chosen to make the special status of these directories clear in a check-out structure, and the RELEASES directory has been added.

Further directory structure is built under the TRUNK directory. The following top-level division may be used:

- **apps**: subdirectories contain sources corresponding to various standalone applications;
- **components**: subdirectories contain sources corresponding to re-usable components, usually in the form of libraries such as dlls etc;
- **libraries**: subdirectories contain sources of libraries which are (re-)used at compile time by other libraries, components or applications;
- **distrib**: sources or files requred for building or configuring distributable artefacts.

Below this level, directories would normaly have the name of the particular library, component or application. Below that, the following standard names can be used where sensible:

- **src**: source;
- **doc**: documentation of source code;
- **lib**: sources which correspond to a binary library, i.e. a .so or .dll;
- **conf** or **etc**: configuration files;
- **app**: source corresponding to a standalone application or utility.

The above structures should be treated as guidelines; recommended directory structures are usually available from the documentation of tools being used.

© 2003-2006 The openEHR Foundation
email: info@openEHR.org web: www.openEHR.org

5 Release Management

5.1 Overview

A named "release" is formally defined as a named list - known as a "baseline" - of the set of controlled items in a repository, and their individual version numbers at the point of time of the release. Releases correspond to the release of major additions in specification or functionality of the total product, and usually occur in coarse-grained time, e.g. every quarter, six months, or year. Every change request that is processed between releases is targetted to a particular release, usually the next one, but not necessarily - the CM system allows multiple future releases to be running at once.

Releases are created in Subversion repositories simply by performing an svn copy operation, which essentially creates a "lazy" copy of the current state of a repository. Inspecting the RELEASES directory in any repository will which release branches have been created; inspecting the TAGS directory will show exactly the release points that have been tagged. For a given release branch, e.g. "release 0.9" of a repository, there may be more than one tag, e.g. "0.9", "0.9.1", "0.9b" and so on.

Each release proceeds through a number of phases. The rules about what kind of changes can be made to the repository during the phases vary, as shown in the following example:

```
phase = development      any change
phase = test             only changes to correct errors or bugs
phase = production       only changes to correct bugs found in use
```

The actual phases used in *open*EHR repositories may vary with the repository. In fact, only the difference between "development" and "after development" phases are made by branching the repositories.

All releases are named "release-XXX", e.g. "release-1.5", where "1.5" is the release identifier. Release identifiers can be any string.

5.2 Release Structure and Branching

The relationship between releases is worth explaining in some detail, since it is the basis of the workflow of any project. FIGURE 4 illustrates a typical workflow. Typical activities are as follows.

- Work starts on the mainline of a repository (in the TRUNK directory), and continues for some time.
- At some point, it is decided that the state of work is stable enough to declare as a named release that could be used outside the development team. What happens at this point is that a logical "branch" is created, representing the named release, while the mainline evolves s usual. With Subversion, a branch is effected simply by an svn copy of the current TRUNK contents to the a named directory in the RELEASES directory.
- The directory hierarchy in RELEASES corresponding to the logical branch is now considered to be limited to that release, i.e. it is considered to be in a testing phase. The only allowable changes to it are bug and documentation fixes.
- Tags can be made of any of the states of a release by performing the appropriate svn copy command from (for example) RELEASES/release-xyz to TAGS/release-xyz.n.
- Meanwhile, general development continues on the repository mainline in the TRUNK. As time goes on, fixes will accumulate in each release branch made to date, due to testing and use. Usually these fixes will be needed on the mainline development as well; the way to obtain them is to create a "patch" containing changes since the beginning of the branch repository, and apply it to the repository mainline (red dashed arrows right to left).

© 2003-2006 The openEHR Foundation
email: info@openEHR.org web: www.openEHR.org

- This operation may be repeated, where each patch is generated from the change following the last change in the previous patch.

Eventually another release will be declared, and the whole operation will repeat, leading to (for example), xxx-dev, xxx-0.9 and xxx-1.0 repositories. As time goes on, users will start using the xxx-1.0 release, and xxx-0.9 will fall into disuse, and could ultimately be declared obsolete (no longer supported) and be archived.

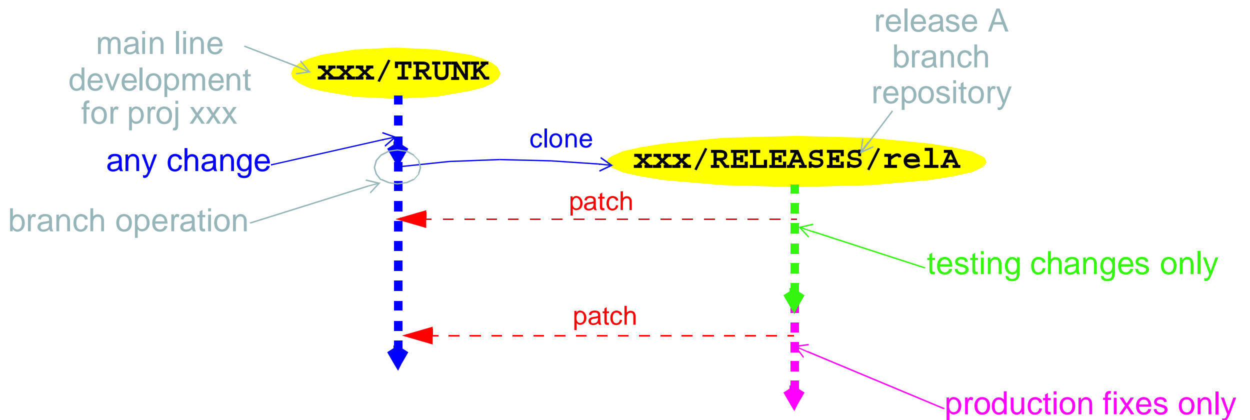

FIGURE 4 Branching

FIGURE 5 illustrates this release logic applied to the *open*EHR specification repository. The initial repository is spec-dev, i.e. the main line of development in which all kinds of changes are added. At some point it will be cloned into spec-0.9, a branch for the 0.9 release of the *open*EHR specifications. The only changes permitted to be done to the spec-0.9 repository are those that fix bugs or problems designated to be fixed in release 0.9. At a later point in time, a spec-1.0 repository is created.

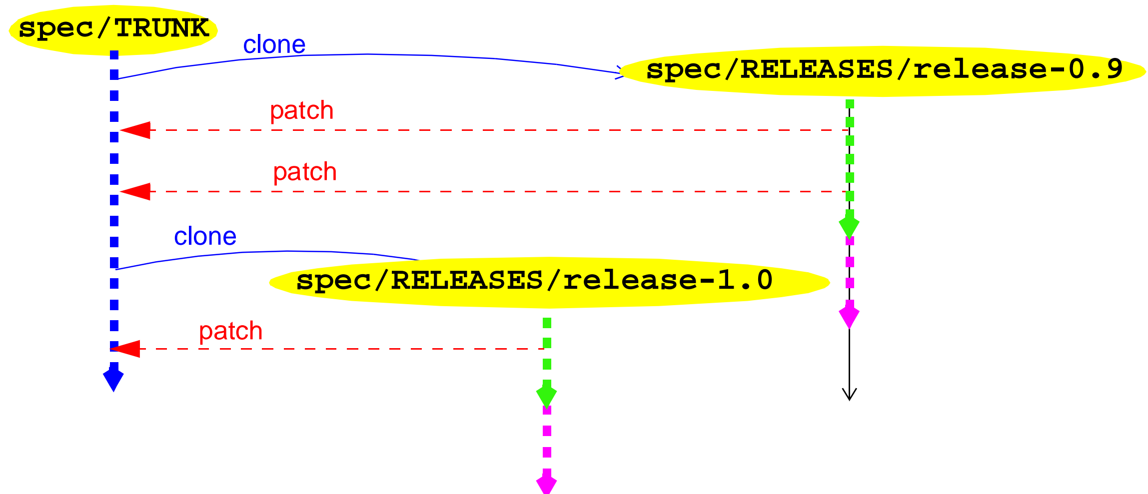

FIGURE 5 Release Structure for the specification repository

The changes made to release-0.9 and release-1.0 may be transmitted one at a time back to specification/TRUNK, by systematic patching, or cumulative patches may be made. Patches may also be made from specification/RELEASES/release-0.9 to specification/RELEASES/release-1.0.

© 2003-2006 The openEHR Foundation
email: info@openEHR.org web: www.openEHR.org

5.3 Release Naming

With the advent of Release 1.0 of openEHR, more rigorous change management rules come into play. These are designed to protect developers and users from adverse effects of changes, and to allow them to upgrade in an orderly fashion. Future releases will follow a 3-digit numbering scheme similar to many open source projects, e.g. Apache, using identifiers like 1.0.1 etc. The meaning of a change in each digit is as follows:

- 3rd position: used to indicate error corrections and minor additions that do not change the semantics. Thus, Release 1.0.2 is the second error correction release after Release 1.0.
- 2nd position: used to indicate significant additions that do not change the semantics of the existing part of the release. Release 1.3.0 would be the 3rd release containing compatible additions to Release 1.0.
- 1st position: used to indicate changes to the semantics or large changes. Release 2.0 would contain changes incompatible in some way with Release 1.0, most likely requiring software upgrade and possibly data migration.

Changes to Documentation

Where changes to documentation are made, e.g. due to a request to clarify an explanation, fix a typographical error, a CR will be raised, and the revision number of the affected document(s) will change, but there will not be a new release number.

Error Corrections

Where the changes made are to correct an error in a model, parsing rules or some other aspect of the formal semantics of the specifications (and possibly also change explanatory text), an error-correction release will be made.

Compatible Additions

Where the changes have the effect of adding a new specification or other artifact which is completely compatible with the current release, an enhancement release is made.

Major Changes

Where changes actually alter semantics of existing artefacts, a new major release is declared. Such changes would normally be grouped into as few such releases as possible.

© 2003-2006 The openEHR Foundation
email: info@openEHR.org web: www.openEHR.org

6 Change Management

6.1 Overview

The approach to change management described here has been developed from change management plans used in a number of industrial contexts. Useful published resources for interested readers include the IEEE standards for configuration management, change management and related issues. A good online resource explaining the concepts at the Technical University of Eindhoven (software engineering home page, CM top page).

FIGURE 6 illustrates the overall *open*EHR change environment. For each project, a repository that is controlled by the configuration management (CM) system, and whose items (documents, source, etc) are created and modified by *project groups* (PGs). The entire *open*EHR community can access the repository for retrieval, or "copy-out". Those developers in identified project groups can perform modifications to the controlled items according to the process described below. All community members can raise Problem Reports, and those members in an *open*EHR team can raise Change Requests.

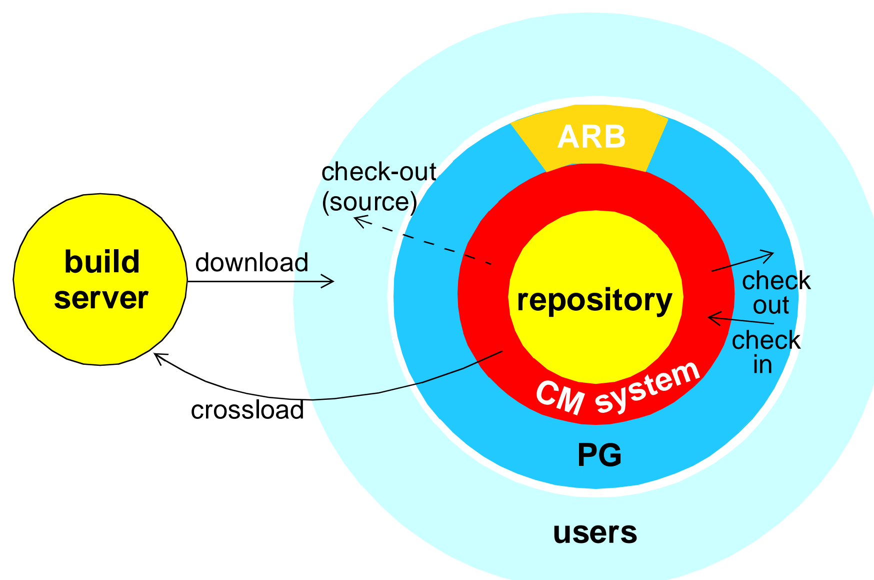

FIGURE 6 *open*EHR change model for reference projects

The key elements of this environment are as follows.

Repository

The repository of deliverables (centre). Includes all documents, software source, and related information needed to recreate a deliverable from scratch.

The Configuration Management (CM) system

The system controlling access to the repository, performs versioning of controlled items, release identification, and manages change requests. The CM system enables any previous version of the repository to be obtained. Implemented in *open*EHR using Subversion and various online change request and problem reporting tools.

Project Group (PG)

Formally constituted team that is responsible for the development of deliverables of the project. These teams can be considered "formal" developers in the sense that they are defined users in the CM

© 2003-2006 The openEHR Foundation
email: info@openEHR.org web: www.openEHR.org

system and can execute a change via a check-out / modify / validate / check-in sequence. The project group can raise Problem Reports (PRs) and Change Requests (CRs) at any time and are also responsible for preliminary review of non-trivial PRs and CRs according to the change management process described below.

The User Community

The *open*EHR community at large, consisting of any user or interested person or organisation. Users in the community who are not in the informal or formal development pool, can copy-out all deliverables and can raise PRs. Users who are not otherwise developers or technically involved in any way typically only download binary software builds.

The Architectural Review Board (ARB)

The ARB is formally constituted of experts from diverse backgrounds, and operates according to the *open*EHR ARB Terms of Reference. Its main activity is the *review* of major CRs, according to the change management process described below. The ARB does not create PRs or CRs, and it does not review PRs, being concerned only with change. (Naturally there may be some members of the ARB who, in their role as a PG member may create PRs and CRs).

6.2 The Change Process

This section describes in detail the change process that applies to *open*EHR projects. However, readers do not need to know all the details to work on a project - the following processes and documentation are generally supported by online tools that ensure that the process is easy to participate in and follow.

6.2.1 Overview

Changes are made to a repository by members of the relevant project group. All changes have a Change Request (CR). **A CR can only be raised by someone in the project group, and is the key document in the change process**, being used to record all status and analysis information relating to the change from its opening to rejection or resolution. CRs are raised either to fix problems, or to perform enhancements to a component. A CR that is designed to fix a problem may refer to existing PRs (usually problems reported in released binaries by users), or the problem may simply be documented in the CR itself (typically the case when a developer finds a problem).

A PR is raised describing in detail a problem or deficiency in a component or product, as perceived by a user (including developers acting as testers). Such descriptions tend to be at a coarse granularity of component or functionality, and only about main releases. A PR can be created by anyone. A PR corresponds to a “black-box” view of the product or component - the raiser doesn’t care how it is implemented, only that it is not working properly. Problem Reports have a simple lifecycle: they are raised, then either rejected or resolved. If not rejected, a PR is always resolved by one or more CRs. A CR may resolve one or more PRs. FIGURE 7 illustrates the relationship between PRs and CRs within the user and developer spaces respectively.

The scope of CR is always a whole repository (i.e. a whole project) even if it changes only a single file. In many cases, a CR causes changes in one project (e.g. the specification project) that will need to be taken into account in another repository (e.g. one of the implementation projects). It is up to the managers of each project to decide on an appropriate moment to incorporate the relevant changes in their repository.

© 2003-2006 The openEHR Foundation
email: info@openEHR.org web: www.openEHR.org

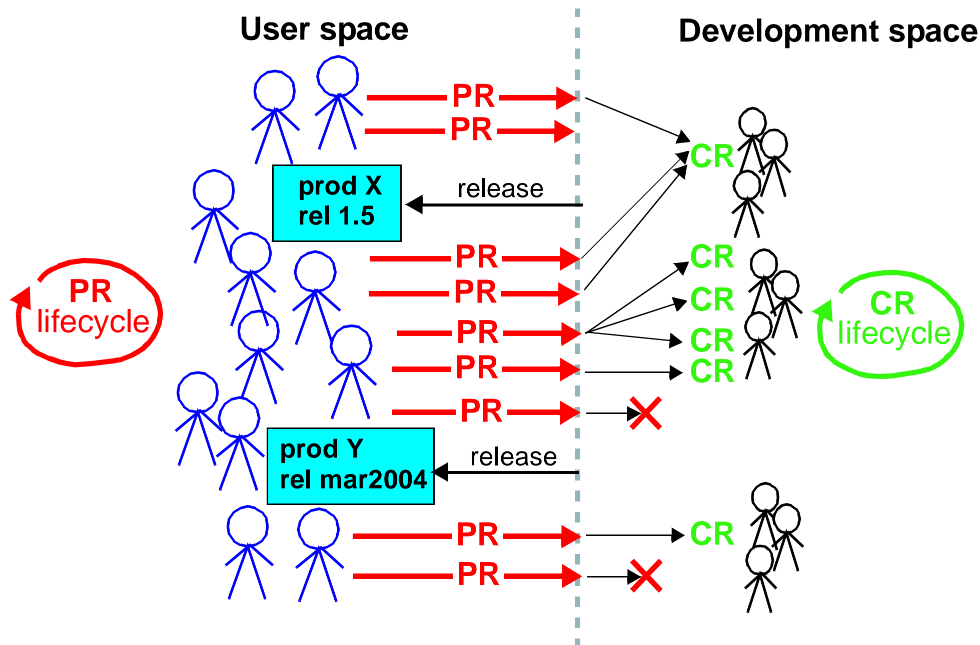

FIGURE 7 Relationship between PRs and CRs

6.2.2 Problem Reporting

New Problem Reports (PRs) are created by users. They are reviewed initially be the relevant project group, and are either rejected or cause the creation of one or more new CRs, or the modification of existing CR(s). Any CR related to a PR in this way should include the PR id in its *problem_description*. The CR then enters the process described below. If the CR is implemented and solves the problem, any PR(s) referred to in its *problem_description* section are progressed to the *resolved* state. FIGURE 8 illustrates the PR lifecycle.

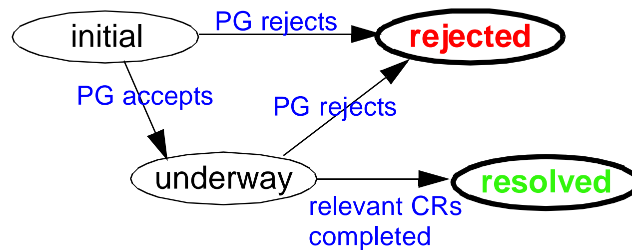

FIGURE 8 PR Lifecycle

6.2.3 Change Request Process for *open*EHR Reference Projects

This section describes a change process in which a board of review as well as the project group processes CRs. In *open*EHR, this process is applied to all reference projects, and any *ad hoc* projects requiring more disciplined change control.

6.2.3.1 Workflow

A new CR created by any project developer, and may be due the review of one or more PRs. The process of handling a CR in a project using a review board is illustrated by FIGURE 9. If a CR is not rejected at some point, it is eventually implemented, causing changes in the appropriate repository.

Assuming sufficient repository-wide quality controls are applied before a CR is closed (such as document review, guaranteeing that change source compiles, builds and runs, and so on), the repository is

© 2003-2006 The openEHR Foundation
email: info@openEHR.org web: www.openEHR.org

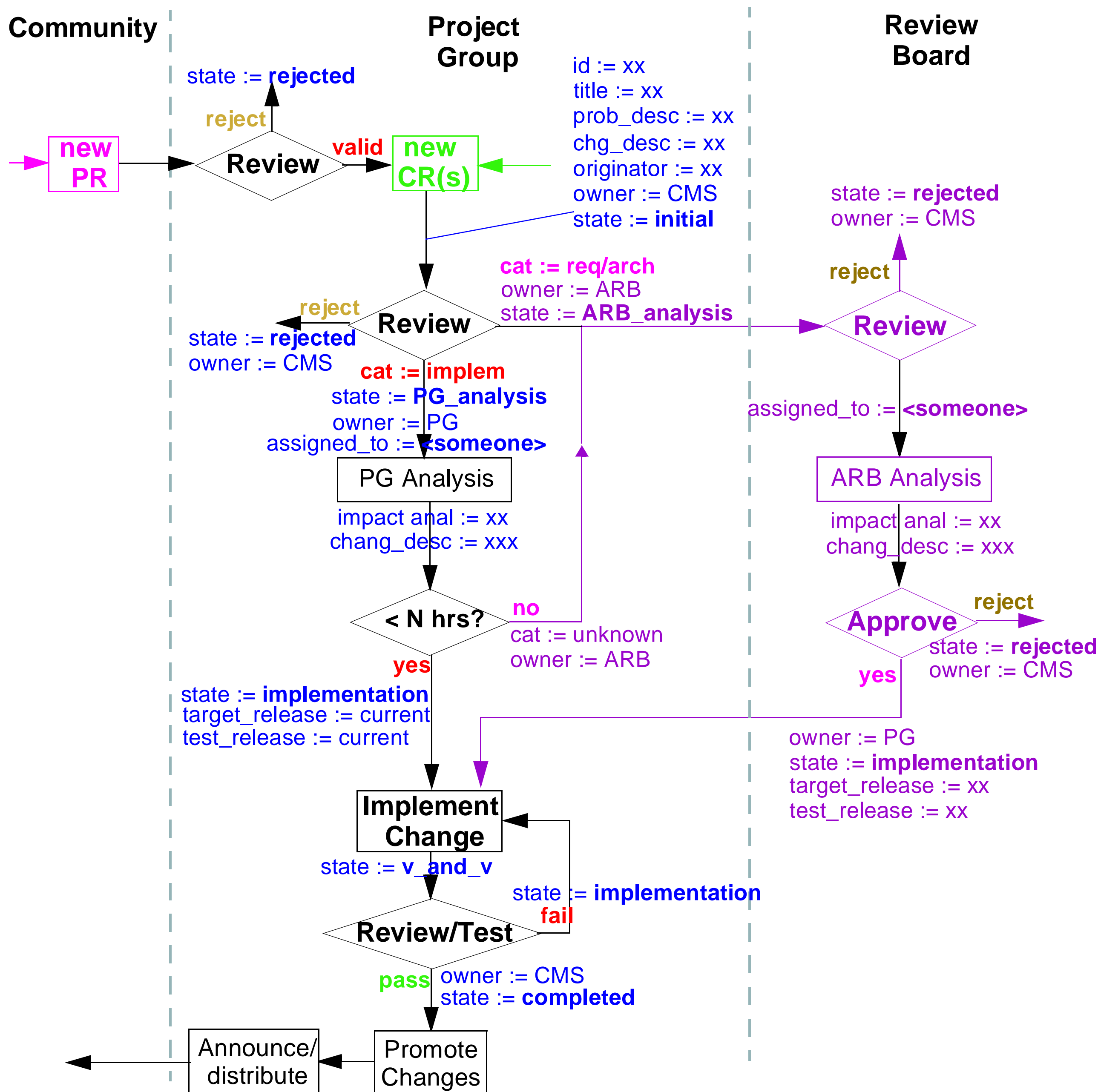

FIGURE 9 *open*EHR Change Request Process for reference project

always guaranteed to move from one self-consistent state to another self-consistent state - there are no inconsistent states. This also means that every version of the repository is the product of some initial state plus the application of a known list of CRs. In this way, the quality of the repository is maintained.

6.2.3.2 CR Lifecycle

Each CR follows the lifecycle illustrated in FIGURE 10. The lifecycle is effectively the CR-centric view of what happens during the change process shown in FIGURE 9. Most CRs proceed through the states *initial*, *analysis*, *implementation*, *v_and_v* ("verification and validation") to *completed*. On the way, some may be *rejected*; others may require ARB analysis and approval before being allowed to proceed. Some CRs may be discovered to be unimplementable during implementation, which will lead to them being put back in the analysis state, from which they might be rejected. Occasionally a

© 2003-2006 The openEHR Foundation
email: info@openEHR.org web: www.openEHR.org

CR may be *superseded* by a more recent decision; this is reflected in the transitions leading to the superseded state. The V&V state is so named because it covers both the testing software and reviewing documents.

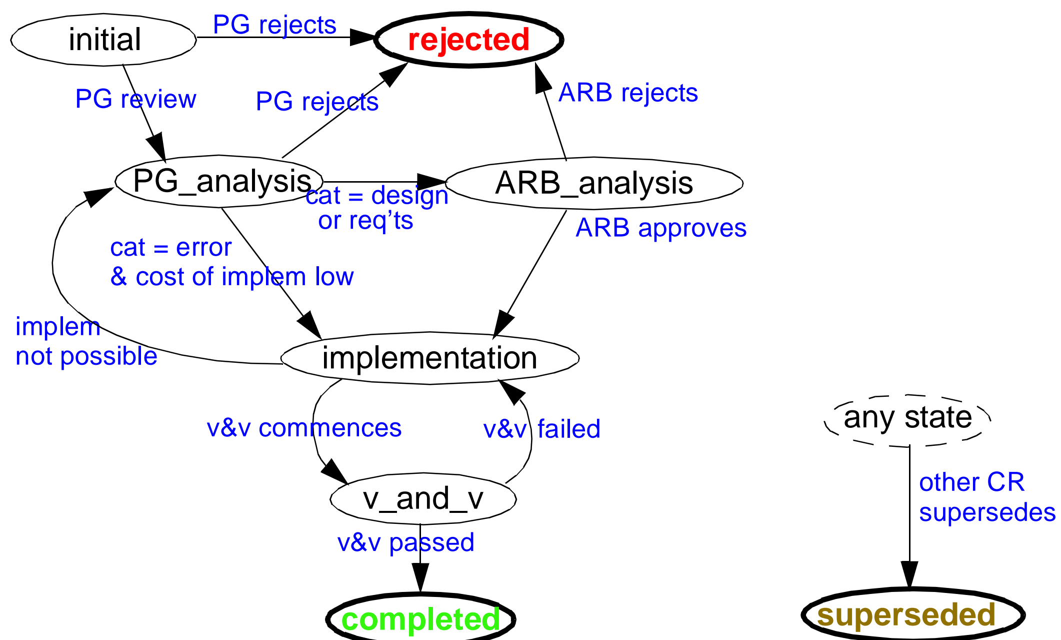

FIGURE 10 CR Lifecycle

When it is decided that a CR will be progressed, it also has to be decided which release the final changes are intended for, and in which release the changes will be made for testing. If the former is production release 1.5, the latter will be the corresponding development release, which must be in test-only mode.

6.2.3.3 Project Group-managed CRs

Many CRs are for trivial problems such as errors in documentation, incorrectly defined elements of specifications, or small software bugs. The CRs are managed by the relevant project group. The PG assigns the CR to someone (or someone self-nominates). **All further changes to the CR are undertaken by the person to whom the CR is assigned to**. The CR is analysed, and if the work to execute it is within the current resources of the PG, it can be carried out. If the work is greater, or it is realised that it is a more serious category of change, the CR is passed to the ARB, by setting *owner*=ARB. For CRs that remain with the project group, the process is as follows:

- Implementation is done in the test release indicated in the CR; when deemed complete, the state is set to *v_and_v*, and the changes are tested/reviewed. If rejected, the CR *state* reverts to *implementation*, and further changes are made, according to the *test_outcome* field.
- When implementation and verirification is complete, the changes are promoted into the repository of the target release indicated in the CR.

6.2.3.4 Review Board-managed CRs

Any CR whose category is requirements or architecture, or for which the work to do the change is significant, is reviewed by the ARB. The review process is as follows:

- a CR normally has an initial *problem description* (that may refer to one or more PRs) and *change description*; there may also be the beginnings of an *impact analysis*

© 2003-2006 The openEHR Foundation
email: info@openEHR.org web: www.openEHR.org

- the CR goes to the ARB with *owner*=ARB and *state*=analysis
- the ARB *assgins* from among its members someone to manage the CR. This person becomes reponsible for progressing the CR through its lifecycle. **All further changes to the CR are undertaken by the CR's *assignee***.
- the ARB members review the CR, and propose changes to the *change_description*, *impact_analysis*, *target_release* and *test_release*. An estimate of time & resources for the work is done, either by the ARB, or by asking a relevant non-ARB person.
- The CR assignee makes changes based on the input, and sets the state to *awaiting_approval*;
- The ARB either:
  - approves the CR (by simple majority vote), in which case it is implemented, tested and the changes incorporated into the relevant repository; or
  - it proposes further changes. Such changes might include setting the target and/or test releases to be some later release, or an experimental one, in order to remove risk to established deliverables; or
  - it discovers that it cannot reach a consensus on the proposed changes as documented in the *change_description* (e.g. there might be a modelling issue) or *impact_analysis* (there might be disagreement on how the change will affect real systems). In this case, the ARB agrees to:
    - hold a physical meeting or telephone conference to resolve the issue;
    - co-opt expert assistance;
    - seek input from the community input
- The CR assignee is responsible for ensuring that by one means or another, the CR is progressed, either to the point where it will be implemented in some release, or else it is rejected.
- When it has been decided that implementation will occur, the *owner* field will be set to PG and the *state* to implementation. In the case of documents or specifications, this simply means that the changes will be made to the documents.
- Implementation is done by the relevant project group in the test release indicated in the CR; when deemed complete, the state is set to *v_and_v*, and the changes are reviewed by the ARB. If rejected, the CR *state* reverts to implementation, and further changes are made, according to the *test_outcome* field.
- When implementation and verirification is complete, the changes are promoted into the repository of the target release indicated in the CR.

Note that CRs are visible to the openEHR community for open inspection online, with an indication of intended dates of resolution & how to communicate a response to the ARB.

6.2.4 Change Request Process for *ad hoc* Projects

This section describes a change process that is used on non-reference (typically smaller) projects, where no formal board of review exists.

6.2.4.1 Workflow

A new CR created by any project developer, and may be due the review of one or more PRs. The process of handling a CR in a project with no review board is illustrated by FIGURE 9. If a CR is not rejected at some point, it is eventually implemented, causing changes in the appropriate repository.

© 2003-2006 The openEHR Foundation
email: info@openEHR.org web: www.openEHR.org

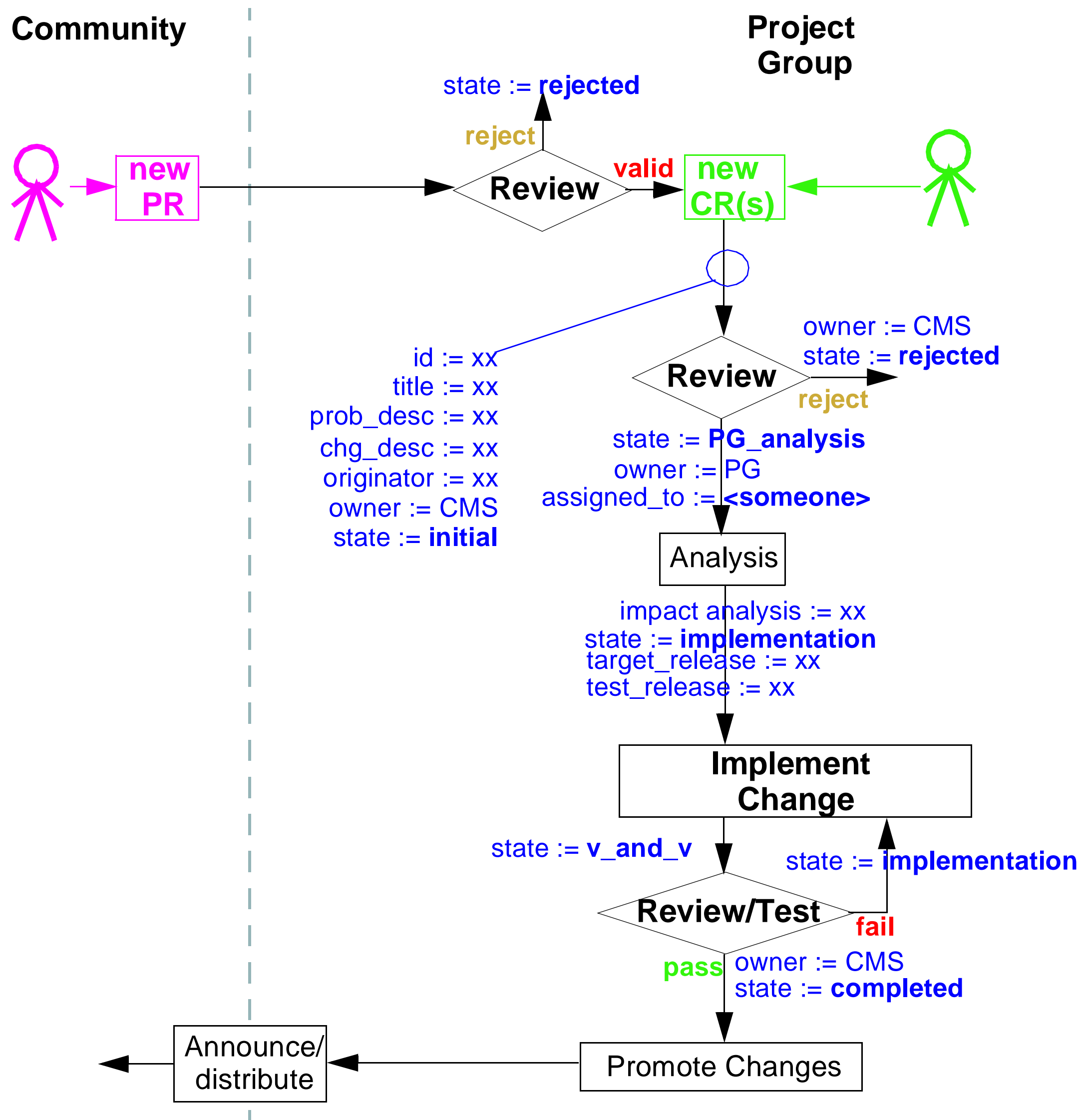

FIGURE 11 *open*EHR Change Request Process with no review board

6.2.4.2 CR Lifecycle

The CR lifecycle, illustrated in FIGURE 12, is similar to the review board case, except that all decisions are taken by the project group (PG).

6.2.4.3 CR Management

CRs in the no review board situation are all managed by the project group, using the same steps undetaken by the PG and ARB in the review board case.

© 2003-2006 The openEHR Foundation
email: info@openEHR.org web: www.openEHR.org

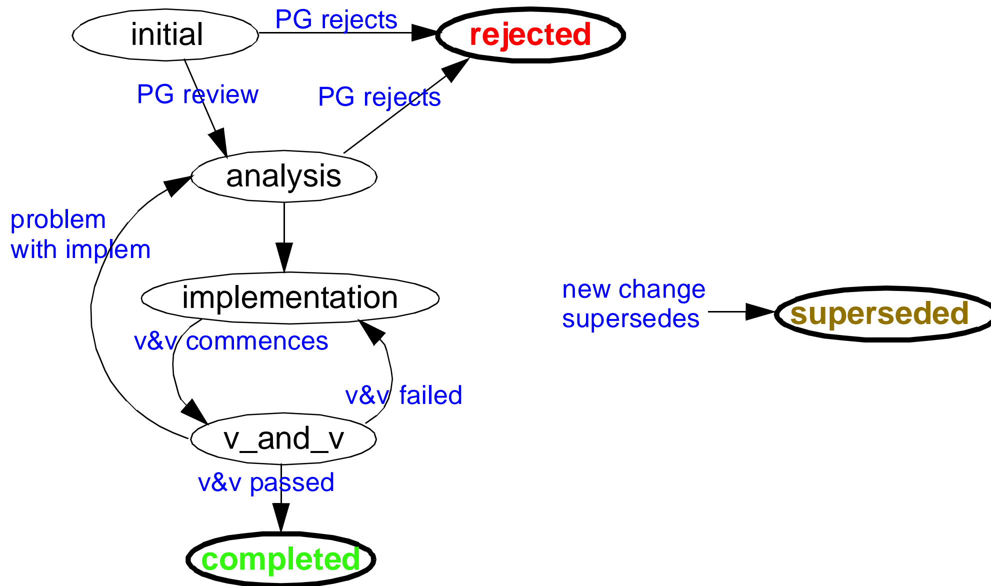

FIGURE 12 Simple CR Lifecycle

© 2003-2006 The openEHR Foundation
email: info@openEHR.org web: www.openEHR.org

7 Tools

7.1 Overview

The following figure illustrates the tool environment supporting *open*EHR development projects.

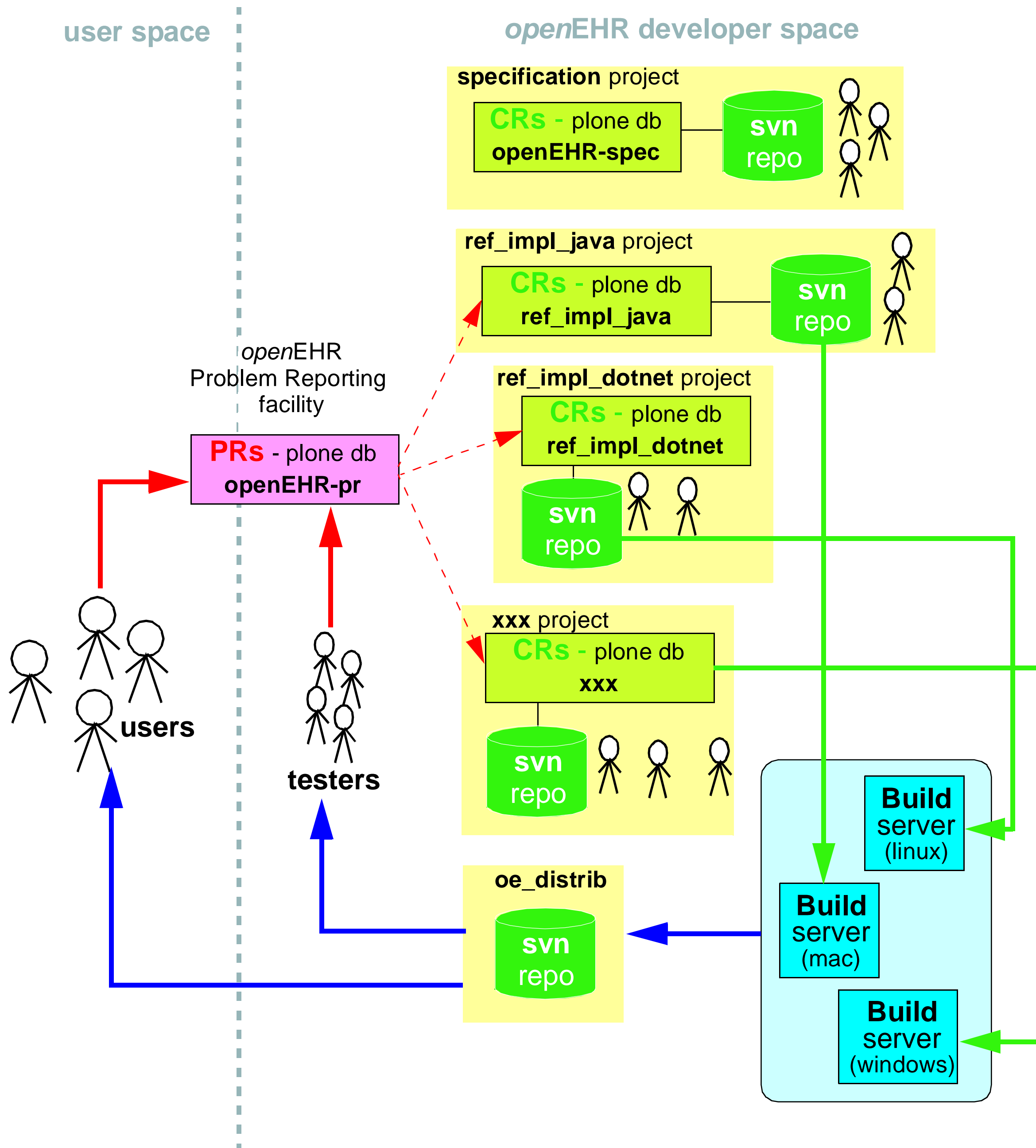

FIGURE 13 Tool Environment of *open*EHR projects

7.2 Configuration Management System

The CM system is the primary tool that supports the change process. *open*EHR uses Subversion for configuration management. [Detailed instructions] for Subversion appear under the "development" link on the *open*EHR website. The master repositories for the *open*EHR projects are hosted on the main *open*EHR server at the [CHIME department] of University College London..

© 2003-2006 The openEHR Foundation
email: info@openEHR.org web: www.openEHR.org

7.3 PR / CR Database

PRs and CRs are managed on a project by project basis, using customised instances of the Plone Issue Collector in a Plone/Zope database at *open*EHR.org.

7.4 Publishing/Distribution

Publishing of usable software packages, documents etc is acheived via the use of the Subversion repository oe_distrib. Its contents are the binary packages for each platform, as of each release and/or tag. The client / server nature of Subversion makes it easy for a user to obtain just what he/she wants, without being concerned with the fact that the entire repository might grow quite large.

© 2003-2006 The openEHR Foundation
email: info@openEHR.org web: www.openEHR.org

8 CI Identification

Configuration items in *open*EHR repositories are generally known simply by their path. The CM tools automatically generate unique, immutable identifiers, which are reliable, regardless of where the file might be moved.

Documents

Documents can be identified using an identifier of the following form:

```
document_id = <artifact_id>
```

Where the fields are defined as:

- *artifact_id*: id corresponding to the subject of document, e.g. “ehr_im” (EHR information model)

Example document identifiers are as follows:

- `ehr_im`
- `common_im`

Document source files will always have names independent of their version, and that might vary according to which tool is used to produce them. Documents may be stored and distributed in various file formats, e.g. Adobe PDF, HTML etc. File names of documents generated for dissemination are of the form:

```
<document_id>.<extension>
```

Where it is more convenient (or it would cause problems with some tools), dots may be replaced by underscores in version numbers. Examples include:

- `ehr_im.pdf` -- Adobe PDF file
- `common_im.html` -- HTML file

Computable Artifacts

All computable artifacts whether abstract or derived are identified by a file name of the form:

```
<artifact_id>.<extension>
```

Neither the project id nor the version id are incorporated in the identifier. The former is redundant in such files, while the latter prevents automatic replacement of a previous version by a later version.

Examples include:

- `ehr_im.idl` -- IDL file
- `common_im.xmi` -- XMI file
- `datatypes_am.xsd` -- X-schema file

Programming language files will almost always be named according to a class name or something similar.

© 2003-2006 The openEHR Foundation
email: info@openEHR.org web: www.openEHR.org

© 2003-2006 The openEHR Foundation
email: info@openEHR.org web: www.openEHR.org

Appendix A Forms

A.1 Problem Report Form

```
                        openEHR PROBLEM REPORT

ID <pr_id>                                          Date Raised: <date>

<Title>

RAISER: <person>                                  STATE: <state>
                                                  *PRIORITY: <priority>
                                                  SEVERITY: <severity>
[RELATED_CRs: <related CRs> ]

                        PROBLEM DESCRIPTION

COMPONENT:   <component_id>
PROBLEM_DESCRIPTION: <text>

                        RESOLUTION

Date Closed: <date>
RESOLUTION_DESCRIPTION: <text>

NOTES: <notes>
```

A.2 PR Form Fields

Field	Value
pr_id	id of form “PR_nnnnnn”
date_raised	yyyy-MM-dd
title	text
raiser	name <email address>
state	opened, review, resolved, rejected
priority	Assigned by *open*EHR; values from 1, 2, 3
severity	critical, high, moderate, low
related CRs	List of CR ids for CRs
component_id	identifiers of released component manifesting problem
problem_description	text - problem as perceived by user
date_closed	yyyy-MM-dd
resolution_description	text
notes	text

© 2003-2006 The openEHR Foundation
email: info@openEHR.org web: www.openEHR.org

A.3 Change Request Form

```
                         openEHR CHANGE REQUEST

ID <cr_id>                                          Date Raised: <date>
               <Title>

RAISER: <person>

OWNER: <CMS | PG | ARB>                                 STATE: <state>
ASSIGNED_TO: <person>

PROBLEM DESCRIPTION: [ text | <list of PR ids>]
[Dependencies: <list of CR ids>]

                         CHANGE DESCRIPTION

CATEGORY: <category: documentation | error | design | requirements>

IMPACT ANALYSIS:    <text>
ANALYST:            <person>
CHANGE DESCRIPTION: <text>

COMPONENTS AFFECTED:
       <change_component, version>
       ...

APPROVED by:        <person>
IMPLEMENTOR:        <person>

TARGET Release: <baseline id>

                         VERIFICATION & VALIDATION

TEST Baseline:  <baseline id>
Test Outcome:  <text>

                         RESOLUTION
Date Closed: <date>
[Reason for Rejection: <text>]

NOTES: <notes>
```

A.4 CR Form Fields

Field	Value
id	CR_nnnnnn
date_raised	yyyy-MM-dd
title	text
raiser	name <email address>

© 2003-2006 The openEHR Foundation
email: info@openEHR.org web: www.openEHR.org

Field	Value
owner	CMS (config mgt system), PG (project group), ARB
assigned_to	name of ARB member responsible for progressing the CR
state	initial, rejected, PG_analysis, ARB_analysis, implementation, v_and_v, completed, superseded
problem description	text description, or else reference to list of ids of PRs generating this CR
dependencies	list of ids of CRs whose completion is required for the completion of this CR
category	documentation, error, design, requirements
impact_analysis	text describing impact on rest of release
analyst	name <email address>
change_description	text describing what should be changed
items affected	list of items
approved by	name <email address>
implementor	name <email address>
target release	release by which this CR must be resolved
test baseline	release in which changes due to this CR will first appear for testing
test outcome	either “passed”, or a reason for test failure
date closed	date on which CR was completed
reason for rejection	text

© 2003-2006 The openEHR Foundation
email: info@openEHR.org web: www.openEHR.org

© 2003-2006 The openEHR Foundation
email: info@openEHR.org web: www.openEHR.org

END OF DOCUMENT

© 2003-2006 The openEHR Foundation
email: info@openEHR.org web: www.openEHR.org