*open*EHR Specification Projects Change Management Plan

Editor:{{T Beale}[1]

Revision: 0.5.2

Pages: 31

1. Ocean Informatics Australia

Copyright Notice

Amendment Record

Issue	Details	Who	Completed
0.5.2	Review by David Ingram.	T Beale	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.

Table of Contents

1 Introduction

1.1 Purpose

The purpose of the *open*EHR Change Management Plan is to define the plan, personnel, procedures and tools for managing the *open*EHR specification projects, i.e, projects in the Requirements, Architecture and Implementation spaces, and projects in the Implementation space.

1.2 Audience

1.3 Status

The latest version of this document can be found in PDF and HTML formats at `http://www.openEHR.org/Doc html/Document/roadmap.htm`. New versions are announced on `openehr-announce@openehr.org`.

1.4 Terms and Acronyms

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

2 Overview

The purpose of the *open*EHR Change Management Plan is to define the plan, personnel, procedures and tools for managing *open*EHR technical projects, i.e, projects relating to the Requirements, Architecture, Engineering and Implementation activity areas. These projects have “controlled” deliverables, and a clear problem reporting and change request interface. Projects not covered by this plan include those in the Knowledge, Educational and Publications areas in the Domain space.

The following table shows the main projects covered by this plan. In this table:

- Names of projects covered by this plan are indicated in the third column.
- Release management is applied to the all of the projects covered by this plan; i.e. a “release of *open*EHR” includes a given version of everything in the Requirements, Architecture, Engineering and Implementation projects.
- The Architecture activity area (second major row) is the main driver of major new releases. That is to say, if the abstract models change in any significant way, a new release is declared. The Archietectural Review Board (ARB) decides on new releases. Changes to the requirements base and to the implementation projects (e.g. the release or correction of an XML schema) will also cause releases.

S	Project	Deliverable	Description
Specification	Requirements	**Requirements Base**	Requirements base of *open*EHR - functional requirements, scenarios etc.
		Conformance to Standards	Conformance statements with respect to existing/emerging standards, e.g. TS ISO I8308.
	Architecture	**Design Principles**	Design principles of health information systems and in particular the EHR
		Reference Model	Abstract Information and Service Models for: • EHR • Demographics • Workflow Abstract Information Models for: • EHR_extract • Common • Data Structures • Data Types • Support (low level primitives) Abstract service model for • Archetype repository
		Archetype Language	• Archetype Definition Language (ADL) specification • Template Definition Language (TDL) specification • Archetype Query Language (AQL) specification
		Archetype System	• The *open*EHR Archetype System
	Engineering	**ITS**	Implementation Technology Specifications (e.g. IDL, XML-schema, various programming languages, database schemas) for: • EHR • EHR_extract • Demographics • Workflow • Common • Data Structures • Data Types • Support
		Conformance	Test cases and plans for testing of conformance of systems against Requirements and ITSs.

S	Project	Deliverable	Description
Implementation	Implementation	**Tools**	Open source implementations of: • ADL Archetype validator • ADL engineer workbench • Clinical ADL editor • Template Builder
		Reference Implementation	Open source implementations of: • archetype/reference model validating kernel • EHR server • Demographic server • Workflow server

3 *open*EHR Specification Projects

The *open*EHR *specification* projects are those whose deliverables are considered technical specifications - i.e. that can be used either to develop more concrete specifications, or to build systems, test plans, or other usable artifacts. The specification projects include Requirements, Architecture, and Engineering, the latter of which provides the main usable specifications for

3.1 Requirements Projects

The scope of the requirements projects is defined as being **the set of deliverables which appear in the “Requirements” part of the *open*EHR website**.

3.1.1 Requirements Base

The Requirements Base project aims to develop 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.

3.1.2 Conformance Statements

As EHR specifications emerge from various standards bodies, *open*EHR produces documents describing conformance or other relationships.

3.2 Architecture Projects

The scope of the architecture projects is defined as being **the set of deliverables which appear in the “Architecture” part of the *open*EHR website**. A roadmap document covers the Architectural projects.

3.2.1 Design Principles

The Design Principles project aims to elucidate a set of principles underpinning the architecture of the EHR and related services. It is produced as a document.

3.2.2 Reference Model (RM) Project

The Reference Model Project develops the primary set of *abstract*, formal specifications of *open*EHR systems, defining semantics for information and service interfaces of health information systems. These abstract expressions are independent of implementation technologies, and are distinct from the concrete expressions developed by the ITS Project in the Implementation space. The scope of the RM Project includes, but is not limited to:

- formal specifications for the ISO RM/ODP information and communication viewpoints of the EHR and other relevant services in an EHR information environment, e.g. Demographics, Access Control and Workflow, expressed as documents containing UML and related formalisms;
- tool-based expressions of the abstract models, e.g. XMI files expressing UML models, EBNF specifications, parse source files etc.

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”. Formal abstract expressions are the ultimate “master” models of *open*EHR.

3.2.3 Archetype Languages Project

The Archetype Languages project develops language specifications for archetypes and templates, inluding the Archetype Definition Language (ADL), Template Definition Language (TDL) and Archetype Query Language (AQL). The project deliverables include:

- Documents describing syntax definitions of the languages
- EBNF language specifications
- EBNF syntaxes for path syntax and query language

3.2.4 Archetype Models Project

This project develops the formal, abstract archetype model describing the semantics of archetypes at runtime, as well as the profiles of information models which are archetyped. The latter essentially describes which classes and which attributes are available for archetyping.

3.3 Engineering Projects

Engineering projects create artifacts which represent formal, directly usable expressions of the *open*EHR specifications, and are used to build systems. The scope of the Engineering project space is defined as **the set of deliverables which appear in the "Engineering" part of the *open*EHR website.**

3.3.1 Implementation Technology Specification (ITS) Project

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, ITS's 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.

3.3.2 Conformance Project

This project develops the artifacts used to carry out testing on externally produced *open*EHR-based implementations, including:

- conformance test plans and test cases for validating any externally produced artifact against a relevant RMP specification

- normative input and output data files for regression testing.

4 Implementation Projects

The Implementation projects produce actual tools or systems based on the formal expressions of the specifications and/or artifacts from the Engineering projects. Implementation projects achieve two things. Firstly they act as a test-bed for the Engineering project outputs; secondly, they provide reference implementations which can be used for educational, research, and operational purposes. The scope of the implementation projects is defined as being **the set of deliverables which appear in the "Implementation" part of the *open*EHR website**.

4.1 Tools Project

This project develops various tools:

- any converter which creates a derived artifact from an abstract one
- tools for validating specifications in the RMP
- tools for working with test data or test cases
- archetype and template validation tools

The deliverables include tool documentation, executable tools, source code, test cases and test data.

4.2 Reference Implementation Project

Reference Implementation projects will in general create open source software components or libraries based directly on the specifications. They may also create test applications, e.g. a basic EHR browser.

5 Repository Organisation

For the purposes of organisation in the CM system, all controlled items are arranged in the logical structure described below. If a user checks out or copies out from the CM system, this is the directory structure they will see.

5.1 Overall Design

The overall design of *open*EHR repositories is shown in FIGURE 1.

```
$OPENEHR
   |
   ├ -openEHR
   |       ├──process -- documents describing overall process
   |       ├──project -- documents relating to project management
   |       └──legal -- documents relating to licences, copyright
   ├ -<project repository>
   |       ├──CM -- Configuration Management area; CRs, PRs, users, etc
   |       ├──<project space>
   |       |        └──<project>
   |       |                └──xxxx -- output of this project
   |       └──publishing -- generated output of projects - readable docs, ex
   |                ├──
   |                └──
   └ -<project repository>
           └──etc
```

FIGURE 1 Repository Design

In this structure, there are several repositories. A "repository" is a unit of change management, meaning that it includes groups of content which should be change managed together - i.e. when a change request is raised, it applies to all items in the repository. Repositories are also the unit of management by CM tools.

The repositories are as follows:

- ***open*EHR**: globally applicable information, including legal, project management, general process, and so on;
- **specification**: the three projects *requirements*, *architecture* and *engineering*;
- **implementation**: the implementation project; since this includes software, it is likely to be large in its own right; it will also lag behind the changes which occur in the specification repository

In each repository, the directory structure is driven by the needs of the project spaces included in that repository. Each repository has a CM directory containing its CM plan and its CRs and PRs.

The following figure illustrates how the abstract structure above translates into a concrete directory structure.

```
$OPENEHR
   |
   └ -specification
          ├──CM
          │   ├──PRs
          │   └──CRs
          ├──requirements
          │     ├──requirements base
          │     └──conformance statements
          ├──architecture
          │     ├──design_principles -- Design Principles project
          │     ├──reference_model  -- Reference Model project
          │     ├──archetype_languages  -- Archetype languages project
          │     └──archetype_model -- Archetype model project
          │
          ├──engineering
          │     ├──ITS -- implementation technology specifications
          │     ├──tools
          │     ├──reference_implementations
          │     └──conformance -- conformance specifications
          └──publishing
```

FIGURE 2 Structure of **specification** Repository

The following sections describe the structures of some of the major *open*EHR projects.

5.2 Reference Model Project

The product directory of the RM project in the Architecture space is illustrated in FIGURE 3.

```
reference_model
   │
   ├──ehr
   │   ├──────im
   │   │      └──────<files>
   │   └──────sm
   │          └──────<files>
   ├── demographic
   │   └──────im
   │          └──────...etc...
   ├── etc
   └── computable -- computable expressions of abstract specs
             ├──xmi -- XMI expressions
             │   └──────ehr_im.xmi
             │          ehr_am.xmi
             │          ehr_sm.xmi
             │          ...etc...
             └──xmi -- eiffel validation expression
```

FIGURE 3 Reference Model Directory Structure

5.3 Archetype Languages Project

The product directory of the Archetype Languages project in the Architecture space is illustrated in FIGURE 4.

```
archetype language
   |
   |----ADL
   |     |----specification -- documentary expressions of ADL
   |     |----computable -- computable expressions of ADL
   |----TDL
   |     |----specification -- documentary expressions of TDL
   |     |----computable -- computable expressions of TDL
   |----AQL
         |----specification -- documentary expressions of AQL
         |----computable -- computable expressions of AQL
```

FIGURE 4 Archetype Languages Project Directory Structure

5.4 Archetype Model Project

The product directory of the Archetype Model project is illustrated in FIGURE 5.

```
archetype_model -- all deliverables of the project
   |
   |----profiles -- documentary expressions of abstract specs
   |       |----ehr
   |       |     |-------<files>
   |       |
   |       |----demographic
   |       :       |-------<files>
   |       :
   |
   |----computable -- computable expressions of abstract specs
           |
           |----xmi -- XMI expressions
                 |-------ehr_im.xmi
                         ehr_am.xmi
                         ehr_sm.xmi
                         ...etc...
```

FIGURE 5 Archetype Model Directory Structure

5.5 ITS Project

The logical structure of the ITS project is shown in FIGURE 6

```
ITS -- implem technology expressions of all abstract specs
  |
  |-------python
  |          └-----ehr_im.py
  |                demographic_im.py
  |                ...etc...
  |-------java
  |          └-----ehr_im.java
  |                demographic_im.java
  |                ...etc...
  |-------<other PL>
  |          └-----ehr_im.xx
  |                demographic_im.xx
  |                ...etc...
  |
  |-------xml-schema
  |          └-----ehr_im.xsd
  |                ehr_am.xsd
  |                demographic_im.xsd
  |                ...etc...
  |-------idl
  |        └-------ehr_sm.idl
  |                demographic_sm.idl
  |                ...etc...
  |
  └-------<other formalism>
```

FIGURE 6 ITS “product” Directory Structure

6 CI Identification

6.1 Documents

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

```
document_id = <artifact_id>-<revision>
```

Where the fields are defined as:

- *artifact_id*: id corresponding to the subject of document, e.g. “ehr_im” (EHR information model)
- *revision*: revision of document, e.g “1.4.3”

Example document identifiers are as follows:

- `ehr_im-4.2`
- `common_im-1.4.3`

Document File Names

Document source files will always have names independent of their version, and which 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-4_2.pdf	-- Adobe PDF file
• common_im-1_4_3.html	-- HTML file
• design_principles-2_4.fm	-- FrameMaker file

6.2 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.

7 Release Management

7.1 Overview

A named "release" is formally defined as a named list - known as a "baseline" - of the set of deliverables 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 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.

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 follows:

```
phase = development      any change
phase = production       only changes to correct errors or bugs
```

Commercial enterprises using CM often implement further phases - typically at least a "production" phase - which are not currently seen as necessary for *open*EHR, since any software it creates is not strictly "in production". This may change in the future.

7.2 Release Structure and Branching

The relationship between releases is worth explaining in a little detail, since it is the basis of the workflow of the entire development effort. FIGURE 7 illustrates a basic release structure. Work starts on a development repository (xxx-relA-dev), and continues for some time. At some point, it is decided that the state of work is good enough to consider releasing it into use. Before this can happen, a period of testing is required. To allow this, while allowing development to continue, a branch operation is done to create the xxx-relB-dev repository, where development changes are now made. Meanwhile testing is carried out on the xxx-relA-dev repository whose purpose is now limited to changes caused by testing and problem reports. When this repository is deemed stable enough for release, it is branched into a production repository, which represents the current, tested "production" release A. Further changes to xxx-relA-dev are only in response to further testing and/or problems reported. Since any fixes have to be tested, the relevant changes are never created in the production repository, they are carried out in the dev repository, tested, and when passed, the final changes are patched through into the production repository (shown by horizontal red dashed arrows in the diagram).

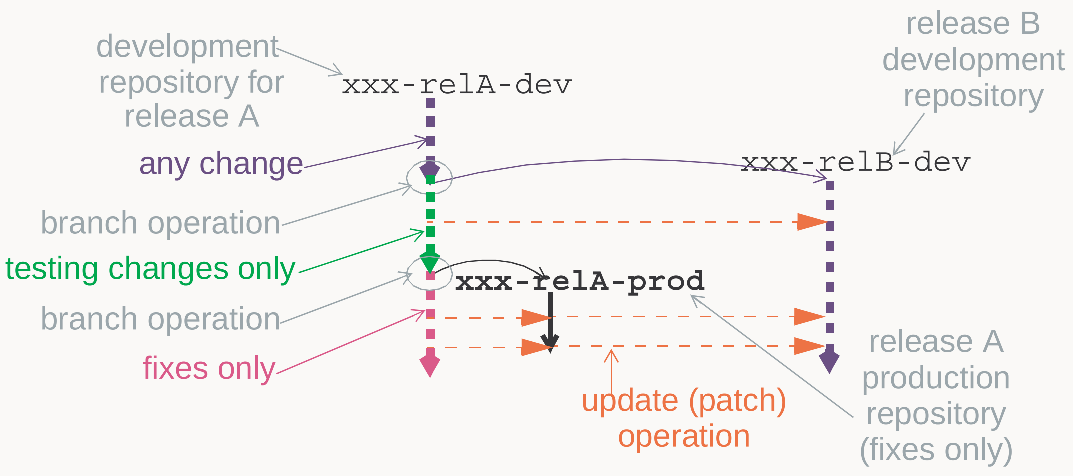

FIGURE 7 Branching

Things continue like this indefinitely, with very old repositories perhaps being decommissioned if there are no longer any users.

FIGURE 8 illustrates this release logic applied to the *open*EHR specification repository. The initial repository is spec-0.9_D, i.e. the development phase of the specification projects. At some point it will be cloned into spec-1.0_D, the development version of release 1.0, and spec-0.9_P, a production version of this release,. The pattern continues, with spec-1.0_D eventually going into production, and also further development. In the example evolution shown below, release 1.5 and 2.0 development lines are created. This is not an official *open*EHR intention; it simply illustrates that multiple development releases could exist at any point in time.

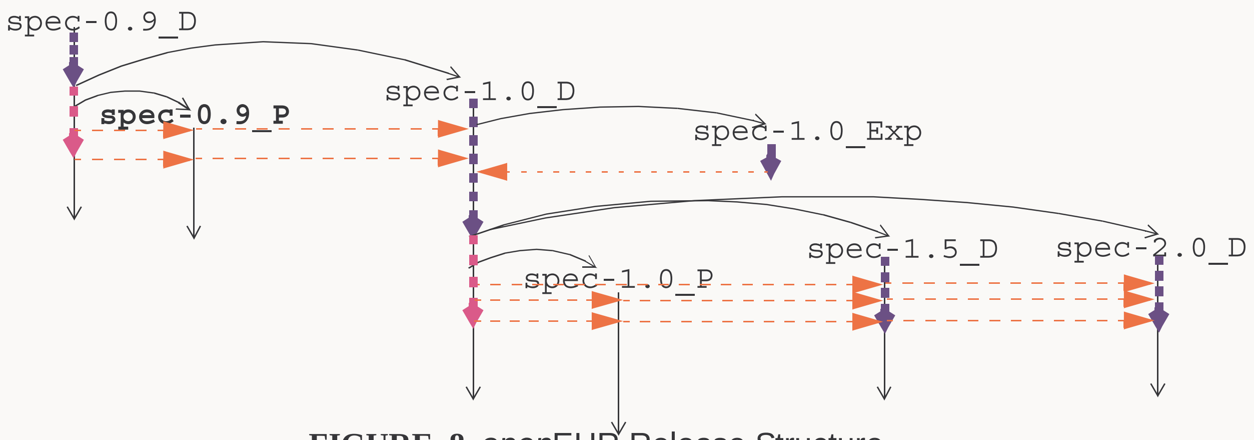

FIGURE 8 *open*EHR Release Structure

Repositories which go into fix-only mode are used to do work only on fixing bugs in code, documentation errors, and testing. When each fix has been approved by the ARB, it will be propagated to all relevant repositories extant at the time, as shown by the horizontal dashed arrows. This might happen numerous times over the life of a recipient repository.

7.3 Interior Versions

Minor releases, i.e. named baselines, can be identified within a major release, as is often done with commercial products. For example, if there was a “jaguar” release of a product, minor releases might be “jaguar A”, or “jaguar Q1 2005” and so on. In *open*EHR, such baselines are named by “tagging” the repository as it is at a point in time, with a name, usually based on the date, such as “july 15 2003”. However, the name could be anything which suits the development process and user community. Baselines are not declared as releases but are otherwise the same, and are known points in the history of the deliverables with which developers always work.

8 Change Management

8.1 Overview

FIGURE 9 illustrates the overall *open*EHR change model graphically. The underlying concept is of a repository which is controlled by a Configuration Management (CM) system, and whose items (documents, source, etc) are created and modified by project development teams. The entire *open*EHR community can access the repository for retrieval. Those developers in identified *project development teams* can perform modifications to the controlled items according to a defined process (described below), which is supported by tools. All community members can raise Problem Reports, and those members engaged in development, either in an *open*EHR team or of any other kind can raise Change Requests.

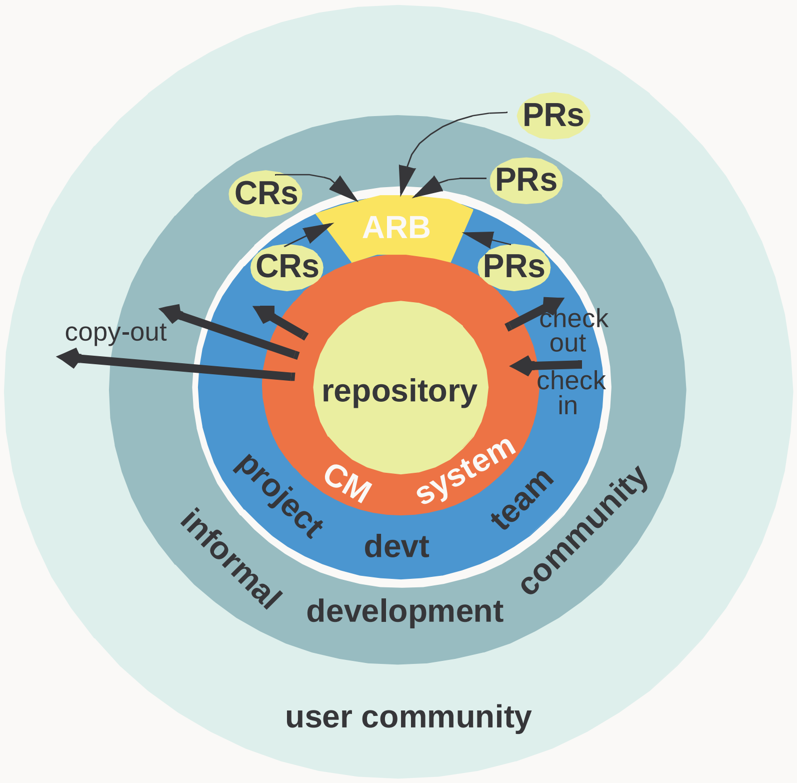

FIGURE 9 *open*EHR Change Model

The key elements of this model are as follows.

Controlled items

The repository of deliverables (centre), known formally as Configuration Items (CIs) in configuration management (CM) theory. Includes all documents, software source, and related information needed to recreate a deliverable from scratch.

The Configuration Management (CM) system

A system which controls access to the repository, versioning of controlled items, release identification, and manages change requests. The CM system enables any previous version of the repository to be obtained.

Project Teams (Formal Development Teams)

Formally constituted teams which are responsible for the development of deliverables of the *open*EHR projects. These teams can be considered “formal” developers in the sense that they are defined users in the CM system and can execute a change via a check-out / modify / validate / check-in sequence. The formal developer team 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 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 activities are the development of new directions for openEHR, and the review of major PRs and CRs, according to the change management process described below.

The Informal Developer Pool

Those members of the community who use the deliverables and develop proposed changes to them. Access to the deliverables is via the logical copy-out operation.

The User Community

The *open*EHR community at large, consisting of any user or interested person or organisation. Users in the community, if they are not in the informal or formal development pool, can copy-out all deliverables and can raise PRs.

8.2 Change Process

8.2.1 Overview

The process of making changes to the repository (including creation of new items) follows a simple model. PRs and CRs are raised by various actors in the openEHR community as follows:

- a PR is raised describing in detail a problem or lack in the deliverables; this is most likely to be written by a user or user organisation;
- a CR is generated in which the problem is summarised in the *problem_description* field of the CR (i.e. no separate PR is written); in this case, the author is most likely to be a developer;

PRs and CRs are reviewed according to the process described below, and are eventually either rejected or implemented. The CR is the key document in the change process, and is used to record all status and analysis information relating to the change. The scope of CR is always a whole repository (e.g. a change to requirements, architectural specification and engineering items) even if it changes only a single file. In many cases, a CR causes changes in one repository - e.g. the specfication repository, which will need to be taken into account in another repository, e.g. the implementation repsository. It is up to the managers of each repository to decide on an appropriate moment to incorporate the relevant changes in their repository. Accordingly, changes in the abstract models will immediately cause changes in the ITSs, since these are all in the specification repository, but it might be some months before the implementation repository is changed to reflect the specification changes.

8.2.2 PR Process

A new PR can be created by anyone. New PRs are reviewed initially be the relevant project group, and are either rejected or cause the creation of a new CR, whose *problem_description* field refers to the PR (and possibly others). This CR then enters the process described below.

8.2.3 CR Process

A new CR created by anyone in the community, or due to the review of a PR has a defined process, illustrated by FIGURE 10. If a CR is not rejected at some point, it is eventually implemented, causing changes in the appropriate repository.

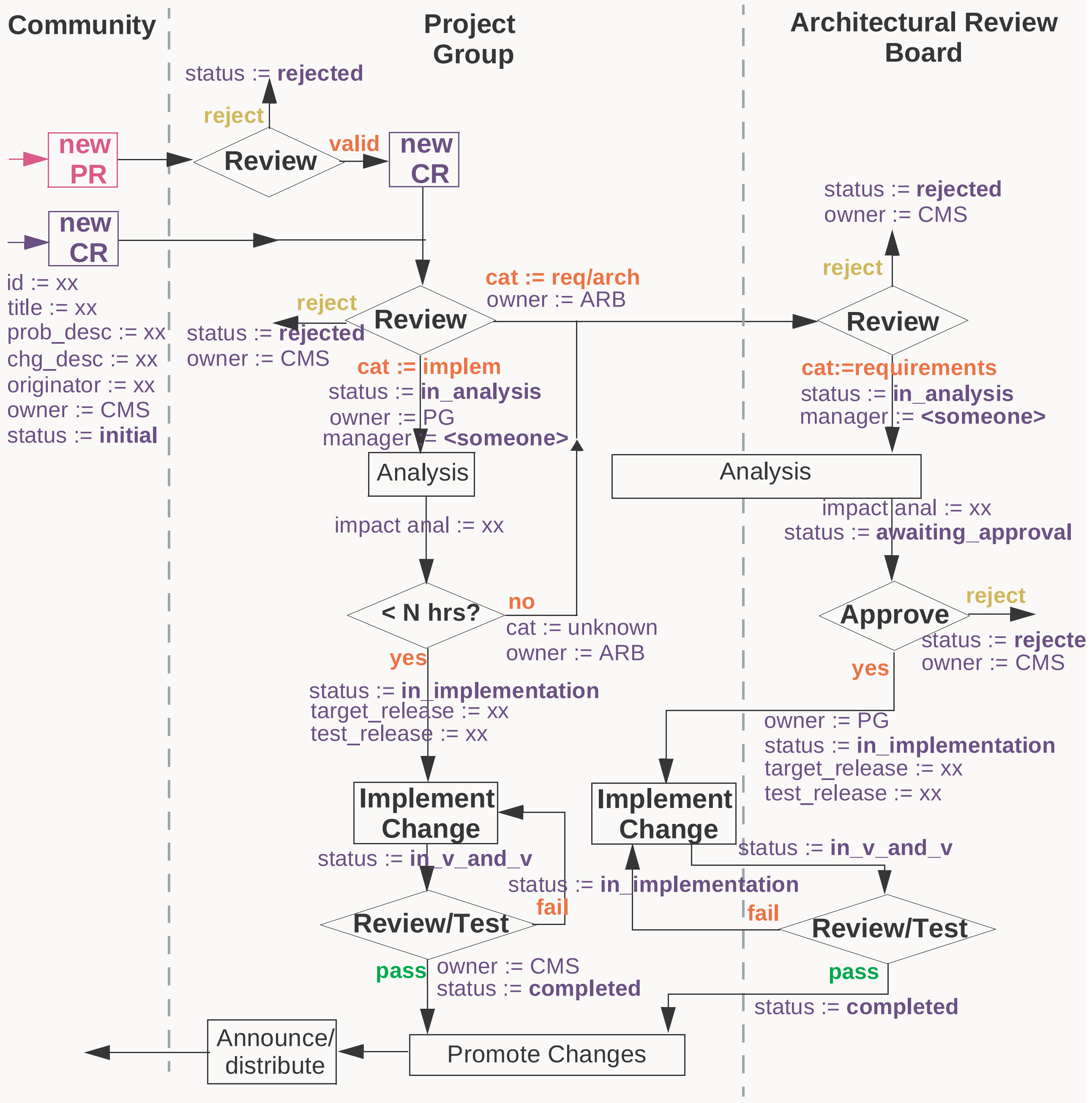

FIGURE 10 *open*EHR Change Request Process

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 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.

8.2.4 CR Lifecycle

Each CR follows a defined lifecycle, illustrated in FIGURE 11. The lifecycle is effectively the CR-centric view of what happens during the change process shown in FIGURE 10. Most CRs proceed through the states *initial*, *in_analysis*, *in_implementation*, *in_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 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.

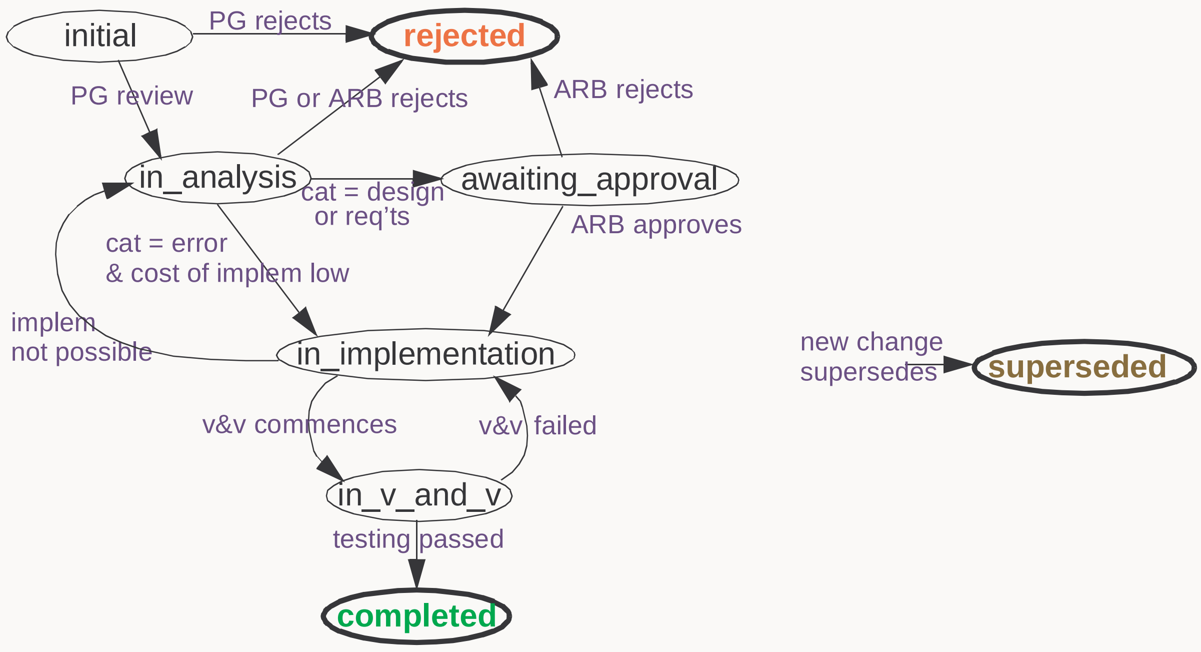

FIGURE 11 CR Lifecycle

When it is decided that a CR will be proceeded with, 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.

8.2.5 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 chooses a manager for the CR (or someone self-nominates). **All further changes to the CR are undertaken by the CR's *manager*.** 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 which 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 status is set to *in_v_and_v*, and the changes are tested/reviewed. If rejected, the CR *status* reverts to in_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.

8.2.6 ARB-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* (which may be an identified PR) and *change description*; there may also be the beginnings of an *impact analysis*
- the CR goes to the ARB with *owner*=ARB and *status*=in_analysis
- the ARB chooses from among its members a *manager* for 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 *manager*.**
- 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 manager 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 manager 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 *status* to in_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 status is set to *in_v_and_v*, and the changes are reviewed by the ARB. If rejected, the CR *status* reverts to in_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.

9 Tools

9.1 Configuration Management System

The CM system is the primary tool which supports the change process.

To Be Continued:

9.1.1 Distributed Working

To Be Continued:

9.2 Publishing Process

To Be Continued:

Appendix A Forms

9.3 Problem Report Form

```
                          openEHR PROBLEM REPORT

ID <pr_id>                                       Date Raised: <date_raised>

<Title>

ORIGINATOR: <originator>                           STATUS: <status>
                                                   *PRIORITY: <priority>
                                                   SEVERITY: <severity>
[RELATED_CRs: <related CRs> ]

                         PROBLEM DESCRIPTION

COMPONENT:   <component_id>
PROBLEM_DESCRIPTION: <problem_description>
[ANALYSIS:   <analysis>]

                         RESOLUTION

Date Closed: <date_closed>
RESOLUTION_DESCRIPTION: <resolution_description>

NOTES: <notes>
```

9.4 PR Form Fields

Field	Value
pr_id	id of form “PR_nnnnnn”
date_raised	yyyy-MM-dd
title	text
originator	name <email address>
status	opened, in_review, resolved, obsolete, 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
analysis	text
date_closed	yyyy-MM-dd
resolution_description	text
notes	text

9.5 Change Request Form

```
                              openEHR CHANGE REQUEST

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

ORIGINATOR: <originator>

OWNER: <owner>                                           STATUS: <status>
MANAGER: <manager>

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

                         CHANGE DESCRIPTION

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

IMPACT ANALYSIS:   <impact_analysis>
ANALYST:           <analyst>
CHANGE DESCRIPTION: <change_description>

COMPONENTS AFFECTED:
       <change_component, version>
       ...

APPROVED by:       <approved_by>
IMPLEMENTOR:       <implementor>

TARGET Release: <target_release>

                         VERIFICATION & VALIDATION

TEST Baseline:  <test_release>
Test Outcome:  <test_outcome>

                         RESOLUTION
Date Closed: <date_closed>
[Reason for Rejection: <reason_for_rejection>]

NOTES: <notes>
```

9.6 CR Form Fields

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

Field	Value
owner	CMS (config mgt system), PG (project group), ARB
manager	name of ARB member responsible for progressing the CR
status	initial, rejected, in_analysis, in_implementation, in_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

END OF DOCUMENT