Project no. 034595

SELF
Science, Education and Learning in Freedom

Instrument: SSA - Specific Support Action
Thematic Priority: IST-2005-2.5.5 – Software and Services

IR8.1: Quality Assessment procedures and mechanisms
Work Package 8

Due date of deliverable: D8, June 2008
Actual submission date: June 2008

Start date of project:
July 1st, 2006

Duration:
2 years

Fundación Vía Libre
Federico Heinz

Revision [**final**]

Project co-funded by the European Commission
within the Sixth Framework Programme (2002-2006)

Dissemination Level

PU	Public	**PU**
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Table of Contents

List of Figures

1 Acknowledgements

This document has been written by Federico Heinz with the help of Beatriz Busaniche and the valuable contributions of the SELF team, especially Rosanna Forestello, Wouter Tebbens and Nagarjuna G. All participants are gratefully thanked for their valuable inputs.

2 About this document

This document describes the procedures and mechanisms that comprise the SELF platform's approach to Quality Assurance. It describes the specific problems that arise when attempting to ensure the quality of collectively-produced materials, and a set of mechanisms that can be put in place to turn those very problems into sources of information on the material's quality and evolution, as well as on the trustworthiness its authors and their level of expertise in specific areas of knowledge.

3 Introduction

This section describes the issues we have taken into account in order to define the mechanisms and procedures for Quality Assurance (QA) in the SELF platform. First of all, we must provide an adequate definition of the term “Quality Assurance,” in order to be able to analyze how it applies to Learning Objects (LOs) and to the process of their cooperative development. This will allow us to identify methods that are well suited to the SELF environment.

3.1 Quality Assurance

The International Organisation for Standardisation (ISO) defines Quality Assurance as “the assembly of all planned and systematic actions necessary to provide adequate confidence that a product, process, or service will satisfy given quality requirements.” In the case of SELF, the products for which we want to ensure that it is of good enough quality are Learning Objects. For the purpose of this project, an LO is defined as a digital entity used in the SELF learning contents. This is in contrast to the definition of the IEEE LOM, whose wider scope includes any entity, digital or non-digital, that may be used for learning, education or training. A course, a lesson, a chapter, an image, an audio file are examples of learning objects in the SELF Platform. In principle, all LOs will be stored as independent files in the SELF repository.

This definition makes it clear that it's the quality of the materials we are concerned with, and not that of the content delivery platform or the actual courses delivered using them, since SELF has neither control nor preferences regarding these issues.

3.2 The problem with “assurance”

SELF's approach to QA must take into account the project's aim of becoming a community-driven platform with low participation threshold for all people involved. The social dynamics of community projects usually dictate that the participation threshold is proportional to the formality of processes within the community. A heavily-formalized project such as Debian, for example, requires that new members of the community actually prove their commitment and technical prowess before they are accepted.

In the case of Debian, this works because of the project's high profile and complexity. It is dubious that new projects like SELF can get away with restrictive rules on who can join before they have achieved a critical membership mass. This severely limits the range and depth of “planned and systematic actions” that can be put in place in the platform.

This is compounded with the fact that, while Debian is essentially a publishing project (i.e. it takes already existing free software programs and publishes them in a coherent fashion), SELF aims also to be a development platform, i.e. a

resource for people who want to work collectively in works in progress. This presents a very particular challenge, which is not present in most processes to which QA mechanisms are applied: we must do it in a way that doesn't prevent incomplete works-in-progress from being published and found on the platform.

Another aspect that differentiates SELF from Debian is that of plurality. Debian's constituency determines what goes in the distribution, and what stays out. This doesn't present much problems in the case of a free software distribution, but when we are talking about learning materials, leaving the editorial decisions to a clique may lead to undesirable effects, such as the suppression of unpopular views. In this context, publishing too much, even to the extreme of publishing contradictory views, is better than publishing too little: we can learn from mistakes only if they are exposed. If they are hidden, we may unnecessarily encounter them again.

All these considerations run in direct conflict with the idea of quality assurance: if the delivered content's quality must be assured, then no contribution can be accepted until its quality has been approved. There's also the issue that every time a contribution is rejected due to quality reasons, it is likely that the author will protest the decision. Regardless of the merits of the protest, experience with other community projects show that just dealing with it can divert significant resources from useful work. It can also be argued that the idea of “planned and systematic actions” is contrary to the spirit of cooperative content production done by loosely-coordinated global groups of people.

3.3 Quality Assessment as a community-friendly approach to QA

While quality assurance procedures are at odds with some of the goals that are crucial to SELF's success, there is an alternative, less restrictive approach that avoids the conflicts while providing equivalent value and even solving some other problems in the process: allowing all materials to be published, while providing the user with the tools to perform community-based, peer-to-peer quality assessment and to filter out those materials which don't satisfy his or her quality standards.

This mechanisms provides the benefits of quality assurance (i.e. being able to trust the contents of the learning objects) without restricting the publication of works in progress or imposing a rigid structure on the production process. The focus of SELF's QA approach will thus not be on quality assurance methods and concepts, but rather on enabling the system to accurately assess each learning object's quality, and on helping users to identify not just high quality materials, but also promising ones which may need help to get better.

4 QA Mechanisms in the SELF platform

A key feature of the SELF platform, which makes it possible for distributed quality assessment to effectively do the job of traditional, process-oriented quality assurance is the fact it does not just publish learning objects, but rather the whole history of each LO. Not just the latest version, but all existing versions, including all intermediate steps and even failed experiments. It is thus meaningless to say that a given LO is of good or bad quality: we must identify which version of the LO we are talking about. The quality is thus not associated to each LO, but to each version of each LO.

The job at hand, then is to help users collectively assess the quality of each version of each LO, by giving them feedback on key metrics on every one of them. This approach not only provides the same benefit to users that quality assurance does, it also solves a number of other problems in the process.

4.1 QA as a distributed community task

Although SELF is focused on free software and open standards, this field of knowledge is broad enough that setting up and running expert groups with sufficient diversity to ensure the quality of published materials becomes a daunting task for any organization. Getting approval from such a body for possible thousands of documents quickly becomes unmanageable.

The standard technique in the free software world for dealing with unmanageable tasks is to leave the task to the community. The decision to expose all of each document's history to the users, and to let the community identify the worthy versions, removes the need for a decision on each publication and allows the project to offload the task of evaluating them to a wide audience of specialists: the platform's users themselves.

For the method to be effective, however, it is important that users understand its mechanics and benefits. This is a case in which bona fide users who don't use the system properly are more dangerous than malicious ones: if most legitimate users adhere to the system's policy, malicious activity becomes evident and can be routed around rather easily. It is thus important to educate users on the proper use of SELF's QA mechanisms.

The community-based quality assessment mechanism is structured around two main metrics: LO popularity (“perceived quality”) and author reputation (“expected quality”). The former is measured in terms of both the number and the reputation of people who have enough regard for the quality of a given version of an LO to keep track of its evolution through their bookshelves. This allows people to assess how each version of each LO is regarded by people who work on that area.

QA Metrics	
Popularity	Absolute number of times the LO is in a bookshelf, either directly or as part of a composite LO.
Length of edit chain	Length of the chain of edits leading to the version. Since each edit validates the parts that don't change, the length of the chain validates the surviving text
# of authors, weighed by reputation	Just the sum of the reputations of the authors involved
# of page views	Indication of how often the page has been read (this is an optional metric, as viewing a page is more of an indicator of interest in the subject in general than in the LO in particular)
Reputation	
Autor reputation	for each version of each LO to which the author has contributed, multiply its popularity by the author's percentual contribution, and add them all together, then divide by the highest absolute score achieved so far, in order to get a number in the [0..1] interval.

Figure 1: QA-related metrics

Authors are also rated according to the popularity of their contributions, which is apportioned to them proportionately to their involvement. The popularity of each author influences the way the system evaluates quality of each version in two different ways: for works in which the author participates, her reputation adds to the “expected quality” measure, in consonance with the generally accepted principle that “good materials are produced by good authors”. This allows for promising materials to be identified early when reputable authors begin collaborating with it.

For works in which the author does not participate, on the other hand, her reputation acts as a weight factor on the popularity, or “perceived quality” factor: the boost each version gets for being on the shelf of a user is proportional to the user's reputation.

4.2 The Bookshelf

The platform provides a working area for users to store references to the LOs they are working on, known as “the bookshelf”. The bookshelf is an important component of the quality assessment mechanism, and has several features that help keep the feedback loop running.

1. Each bookshelf slot does not reference an LO, but a specific version of a Learning Object.
2. The bookshelf keeps users informed when newer versions of the LO have been published, so they can take a decision whether to update their reference to a newer version or not.

3. The system keeps track of which users keep which versions of which LOs in their bookshelves as a measure of popularity.
4. Bookshelves could, in theory have unlimited capacity. The plaform, however, keeps the number of bookshelf slots limited, in order to make them valuable to users. This way, keeping a version on the bookshelf has a real cost for the user, which makes it likelier that only versions that are held in high esteem remain on it.

Not all bookshelves have the same size or “popularity weight”. Users with a high reputation have larger bookshelves, and the popularity weight each slot contributes to its occupant is larger.

4.3 Avoid politics

Many community projects are riddled with political struggle. This is basically a consequence of the fact that those projects give a special status to certain products over others. In Wikipedia, for instance, the last version of an article supersedes any older versions, and although the user can choose to see the history, this requires an extra action. Furthermore, Wikipedia only knows how to handle a linear history, and cannot cope with diverging branches of an article.

The competition for this position of privilege leads to conflicts, and to the so-called “changefests”: a small number of users who engage in a rapid series of changes to a small set of pages, each one removing the other's contributions and adding their own, which in turn get quickly replaced. This often leads to “authority figures” being called to the scene to mediate in the conflict.

SELF solves this problem by removing the source of the conflict: there is no designated “official” version of each LO. The SELF platform effectively treats all versions as equals, thus removing politics and quarrelling as an effective way of attaining notoriety for one's own contributions.

The fact that there is no designated “official” version of an LO, and that SELF's view of an LO's history is not linear provides SELF users with a better alternative to changefests: when they don't like a contribution, they can just contribute around it. Instead of modifying the conflictive version, all they have to do is add their contribution to its immediate predecessor, thus creating a new branch. If the shunned version truly does not contribute anything valuable, it will just remain abandoned.

The version may, however, not have been rejected because it was poor quality, but because it expressed a different point of view. Some users may agree with the rejected version, and they are likely to view the new branch as the conflictive one. The good thing is, they are still free to continue building a branch upon whatever version they like best. This means that diverging views or theories can be expressed on the same LO history, and the structure of the history will eventually mirror that of the underlying dispute. This is a better approach to plurality than attempting to enforce a “neutral point of view” of dubious existence.

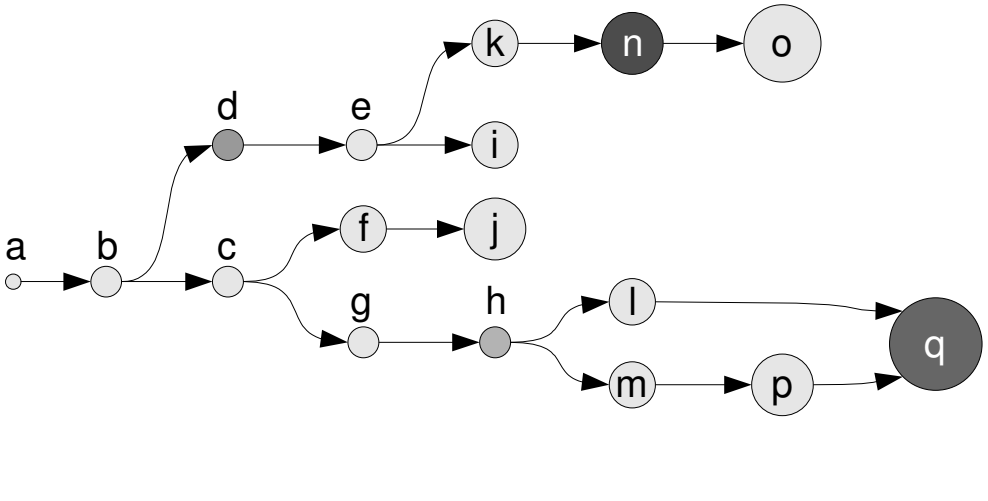

Figure 2: Sample history of an object

The diagram in Figure 2 illustrates a possible history for a learning object. In it, each circle represents a version of the LO. The darkness of the circle indicates the popularity of that particular version (which is measured in-system through mechanisms described in Section 5), the size of the circle is an indication of the number of different people who have contributed to it (in each edit cycle, at most one new contributor can be added).

The diagram tells us a great deal about the LO's history, and even about the subject matter. It is pretty obvious that there are two diverging views on this subject, with a significant amount of people supporting each one of them. The original LO (“a”) started evolving with the contributions of two people, who didn't take long to realize that they don't agree on everything: when one of them contributed version “c”, the other one decided not to build on top of that, but rather on version “b”, creating the branch that starts at “d”. We know it was a disagreement between these two people because the circles don' t get bigger, hence the number of contributors to both “c” and “d” is the same.

The author of the “c” branch continued to build on it with versions “g” and “h”, while other people were busy also building versions “f” and “j” on top of “c”. These versions don' t seem to have attracted much attention: maybe they were very low quality, or represent a fringe point of view within the “c” branch, which continues to grow along another route: three different users built on top of version “h”, in two different branches that eventually were merged into version “q”, which currently is the most popular version on that branch and can be safely assumed to be the best version of those available on the platform for this particular point of view.

Meanwhile, the “d” branch evolved in a rather uneventful fashion, with people becoming involved along the way. Version “i”, was dismissed and abandoned. It was either irrelevant, or it may have been an ill-fated attempt to introduce the “c” branch's point of view into the “d” branch, and supporters of the “d” branch

simply contributed around it.

4.4 Users pick and choose what they use

One of SELF's features is that it is meant to be a useful tool for educators. Thus, its users are mainly people who are fluent in the subject matter they teach, and who can assess each other's work quite effectively. So, instead of using an indicator that is irrelevant to actual quality, such as how recent the article is, or its size, we can rely on our user's ability to tell us which materials are good, and which are bad.

When a user first encounters an LO, she is not presented with any particular version of it, but with a tree representation of the LO's history, showing all its branches and intermediate versions. Users can then inspect the different versions, and use their own judgement and expertise to decide which one they'd rather work on. This removes the emphasis on being the last person to edit a document, and puts it where it belongs: in contributing to produce a document that will appeal to the widest audience.

In a platform that hosts thousands of documents, each one with many different versions, this can be a daunting task, of course, but the system can offer assistance: the system keeps track of each user's actions, and try to infer quality indicators from quantitative, ones such as how many people have contributed to each LO, how many people prefer this learning object over others on the same subject, and also how many people prefer a certain version of an LO over previous and even later versions.

4.5 Modeling vs. correlating

The greatest challenge when assessing the quality of learning objects is finding a meaningful way to aggregate measurements taken along several different dimensions into a some kind of metric to represent the LO's overall quality. As a matter of fact, any formula one may choose to achieve such a goal is disputable, and probably more a result of the proponent's own conceptions than an objective viewpoint on issues such as whether a document with many authors is more likely to be of good quality than one with fewer authors, or whether popularity withing the SELF environment is indeed a good indicator of quality indicator or not.

For this reason, the SELF platform at first will not attempt to do such an aggregation, but rather display a few key metrics for each LO, leaving the task of interpreting them to the user. The next step will certainly involve working together with the community to find useful ways to combine these metrics into something useful, perhaps through the mechanism of user-defined formulas.

The ultimate approach to the problem, however, will be enabled when the document and user base of the SELF platform has become large enough to be able to perform statistical analysis on it. Regardless of whether we attempt to aggregate them or not, the platform will constantly gather information on each

LO's history and on each user's activity. When we arrive at a size large enough, it will be feasible to perform an in-depth quality analysis of a sample of the available learning objects by experts in the respective fields, and then find how the collected metrics on LOs and authors correlate to the way how the experts evaluate them. We expect that this correlations will be more accurate at predicting the quality of an LO (and even at predicting the expected evolution of an LO's quality) than any *a priori* model could be.

5 Catalog of current and proposed QA-related user activities

Most user activities in the system, such as contributing to an LO, keeping track of its development through the Bookshelf, using it in a composite LO, etc, are monitored by the system as indirect indicators of quality. In this sense, pretty much every change a user makes to the platform's database contributes a small amount of information that can be used to infer the quality of the LOs affected by the change. Some special activities, such as rating LO's on specific criteria, are directly related to Quality Assessment.

This section outlines, in the form of “user stories” the different activities a user can engage on which are related to quality assessment. Not all of these activities are implemented in the version of the platform that is available at the time of writing. The most advanced concepts, such as user-provided formulae and explicit user trust are forward-looking ideas to be implemented in more advanced versions of the platform's software.

5.1 Version Control

5.1.1 Selecting a version at a search result

As the result of each search operation, the system delivers a list of learning objects that match the search criteria. Due to the version control system, each entry in that list does not represent a single document, but rather its whole history. When a user clicks on a search result, thus, the user is not taken directly to the LO, but rather to a view of the LO's history.

5.1.2 View an LO's version history

For each LO, the system provides a view that displays its history as a tree of versions. Visual feedback is provided to the user as to the popularity of different branches (for example, through the size of each version's graphical representation, or the font size of its title).

The system also provides information, or means to get information about the version, as well as about the amount of change from each version to the next, and a means for users to inspect the changes made between any two versions, to better assess whether the changes are useful or not.

Each version that is used in composite learning objects is marked in a distinctive way, and a mechanism is provided for the user to see which are those learning objects and to navigate to them.

The system also provides here an interface that allows the user to add any of the displayed versions to the Bookshelf.

Each version that is available in the user's bookshelf is marked in a distinctive

way, and a mechanism is provided for the user to move each bookshelf mark to another version, thus updating the bookshelf marker.

When user Alice updates a bookshelf marker, the system checks the Bookshelf of all users who trust her. If user Bob trusts Alice, and has in his Bookshelf a version of the same LO that is behind the version selected by Alice, Bob's Bookshelf reference to the LO is automatically updated to reflect Alice's selection. This effect of trust is not transitive: if user Charlie trusts Bob, but not Alice, he is unaffected.

5.1.3 Merge two versions of an LO

If a user finds that two branches of an LO both have undergone useful changes, she can use a mechanism provided by the LO version history view to merge the two. The mechanism entails selecting the two versions (say, versions *A* and *B*) and launching the merge function.

The merge function may find that the changes in the two versions conflict with each other (i.e. both versions change the same portion of text in different ways), the system notifies the user of the conflict, and asks the user to resolve them on one of the versions, say on version *A*. The resolution of the conflict creates a new version *A'*, which is then merged with version B to create a new version *C.*

5.1.4 Select a specific version of an LO

When the system needs the user to select a specific version of an LO for a task, it displays the view described in the previous section. The user specifies a version either by clicking on it, or by placing a special marker on it.

5.1.5 Notify the user about available updates to an LO

Whenever a user is looking at a list of learning objects that latches onto a specific version (such as the Bookshelf or a composite LO), the system displays a graphical marker next to each LO for which there are newer versions available. The marker should ideally convey information about how outdated the LO is, by measuring the distance to the head of the longest branch that descends from this version.

An UI element next to the LO (it can be part of the mark displayed to signal the availability of updates) takes the user to the learning object's history, thus allowing the user to select whether to update to a newer version, and to which one.

Upgrading the version of a component of a larger LO creates a new version of the composite LO. In other words: each version of a composite LO is tied to a specific version of each component LO. To change the version of one or more components of a composite LO, the system will create a new version of the composite with the new version references, while the original version will stay as it was before.

5.1.6 Create a new version of an atomic LO

When viewing a list of LOs that includes atomic LOs, be it the Bookshelf, a composite LO, the result of a search, etc, the system provides a mechanism to invoke the text editor on each one of them. The editor allows the user to perform changes to the LO and to save them.

There are two modes of saving the changes: “Save” and “Commit”. When the user clicks on the “Save” option, the changed document is saved to a copy that is private to the user. Other users can't see this version, nor can they refer to it in any way. When the user is viewing one of the previously mentioned lists, the system provides feedback on which of those LOs have been privately changed by the user. Clicking on the edit option opens the editor on the changed version, not on the original one. To edit the original version, the user must first discard the saved changes.

5.1.7 Discard private changes

When the user is viewing any list of LOs, the system provides feedback on which of those LOs have been privately changed by the user, and a mechanism through which the user can revert the LO to the original version, discarding the private changes.

5.1.8 Commit private changes

When the user is satisfied with the changes made to the LO, she can make them public by using the “Commit” mechanism instead of “Save”. This option creates a new version of the LO. If other users have committed changes to the original version of the LO, the version created by the commit option is a sibling of the versions created by other users. If the user wants to integrate the changes to the ones made by other users, she must use the merge mechanism, thus creating yet another version.

5.2 Bookshelf

Each user account has a list of LOs associated with it, called the Bookshelf. The Bookshelf has limited capacity (which can vary with the reputation of the user) and always references a specific version of the LO. Participation in LOs that are present in many Bookshelves has a positive impact on the author's reputation. The Bookshelf can keep references to more than one version of the same LO, each reference takes up one Bookshelf slot.

5.2.1 Adding an LO to the Bookshelf

Whenever a user is looking at a list of learning objects, the system provides visual feedback to identify whether the object is already in the user's Bookshelf. If it is not, it provides a means to add the object to the Bookshelf. If the object

has children (i.e. it is the topmost atom of a LO hierarchy), the mechanism allows the user to choose whether it is just the atom or the composite object that must be stored in the Bookshelf.

The object will only be stored in the Bookshelf if there is a slot available, otherwise the system will ask the user to make room for it. A composite object takes up only one Bookshelf slot.

5.2.2 Removing an LO from the Bookshelf

When the user has the Bookshelf on the screen, the system offers a mechanism to remove each LO from the Bookshelf. Each LO that is removed from the Bookshelf frees up one slot.

5.2.3 Selecting an LO from the Bookshelf

When the user has the Bookshelf on the screen, the system will show a link (which may be anchored on the LO's title itself) that take the user to the editor screen on the object.

5.3 Trusted users

A proposed upgrade to the system involves associating each user account with a list of users that are trusted by the account owner. Being trusted by many users has a positive impact on reputation. The reputation of the trusting users themselves is taken into account to compute how the trusted user's reputation is affected. For the trusting user, versions of materials in which a trusted user participates, or that a trusted user has in his Bookshelf get a boost in their quality rating. Decisions taken by trusted users may affect the trusting user's working environment (for example, LOs in their bookshelves may automatically track the selections of trusted users.). The list of trusted users is of limited capacity (which can vary with the trusting user's reputation).

A possibility to be evaluated is encoding how much a user trusts another one with a real number between -1 (absolute distrust) and +1 (absolute trust).

5.3.1 Viewing the list of trusted users

When logged in, the system provides a user interface element which allows the user to view the list of trusted users. Beyond the usual information and interface elements displayed in user lists, the system provides an interface element to delete the user from the list.

5.3.2 Adding a user to the list of trusted users

When user Alice is viewing user Bob's profile, the system provides an interface element that allows Alice to add Bob to her list of trusted users. If Alice's list of trusted users is full, the system prompts her to remove other users from her list

before she adds Bob to it.

5.4 Manually Rating LOs

Some aspects of LOs cannot be accurately assessed by automatic means, and require feedback by humans.

5.4.1 Rating a version of an LO

When a user is reading an LO, a tab above it allows her to view a series of assertions regarding the LO, and allowing the user to choose a level of agreement to the assertions from the list “disagree strongly / disagree / undecided / agree / agree strongly”. The user may choose to express agreement/ disagreement to any number of these assertions, as well as to leave any question unanswered. The assertions are of the kind “the text is easy to read and understand”, “the text is adequate for the intended audience”, “the text is orthographically and grammatically correct”, etc., which cannot be easily evaluated by automatic means.

The ratings for previous versions of an LO affect subsequent versions of it only if the users who rated the previous versions haven't rated the newer ones yet, and only do so in an attenuated form which gives them less influence with each intervening version.

The influence of the valuations on the quality assessment of the LO will be affected by the reputation of the user doing them. Also, the weight of the user's valuation will be inversely proportional to their authoring participation in the LO.

5.4.2 Re-rating a version of an LO

Each of the user's valuations is recorded as pertinent to the version of the LO that the user is reviewing, and remains connected both to that version and the user. At any time, a user may choose to change the answer to any one of the questions, or even to withdraw any of the answers. If a later version of the object improves/worsens the score on any of the questions, the new valuation must be done in the new version, leaving the earlier version's score untouched.

5.4.3 Entering a QA rating question & answers

The system will provide, for users with administrator privileges, a system to edit QA rating questions. Questions can be either draft (i.e. they are only shown to the author) or published (i.e. they show up in the rating page for LOs). Questions have an “LO type attribute” that allows the system to ask different kinds of questions for different types of media (a question that is relevant to video media may not be useful for textual media).

5.5 Aggregate ratings

The system computes aggregate ratings for each LO, by performing calculations on a number of features, such as the number of Bookshelves the LO is kept in, weighed by the Bookshelf owner's reputation, the reputation of its authors, the number of people who have answered quality-related questions about the LO and the score achieved through these answers, etc.

Since it is unlikely that a single approach will satisfy all users, or that any initial approach will be satisfactory, the system will allow the definition of different formulae to compute the aggregate rating, letting users choose which formula they wish the system to use when it assesses an LO's quality for them.

The system will provide a few pre-defined formulae, one of which is the initial default formula for new users, as well as interfaces to create simple alternative user-defined formulae, which can be published and shared among users.

5.5.1 Selecting a quality assessment formula

The user preferences section of the platform provides a page to select and edit quality assessment formulae. This page lists the available formulae, and can sort and search them according to a series of criteria, such as popularity (how many people use it), author's reputation, author's name, etc. The list includes a description of the formula's intent and other useful information.

For each available formula, the platform provides three basic tools: a way to apply the formula to the LOs in the user's Bookshelf is a useful preview that can give the user an indication of whether the formula matches his preferences or not, a way to select the formula as the quality assessment formula for the user, and way to edit the formula, producing a new version of it.

The ability to edit a formula can be reserved for users with a reputation above a certain threshold.

When the user selects a formula, the selection latches onto the current version.

5.5.2 Editing a formula

The system provides different ways of editing formulae, from a very simple linear combination form in which the user specifies the weight to be assigned to each indicator before their algebraic sum is performed, to a text box in which the user can enter Python code which performs arbitrary computations on data obtained through the platform's API. These availability of different modes of editing can be restricted to users who have reached different reputation thresholds, so as to not overwhelm novice users.

Formulae are version-tracked, just like LOs. When a user produces a new version of a formula, users of the formula are notified and given the choice to update.

5.5.3 Publishing a formula

Users can choose to commit a formula to the world for others to use and evaluate. After they do, the formula becomes available in the list of quality assessment formulae for all users to choose. The number of users of any given formula affects the reputation of its author.