IATI Core Software
NON-FUNCTIONAL REQUIREMENTS

This document defines the non-functional requirements (NFRs) for the production of IATI Core Software. It concerns both the design and development of new components and also instances where existing components are moved to the new integrated platform, as defined by the IATI Technical Stocktake exercise of the summer of 2020.

We are adapting the **Product Quality Model (4.2) of the ISO/IEC 25010:2011 standard** as our NFRs Framework. This model categorises system and software product quality properties into eight characteristics:

1. **Functional suitability**
2. **Performance efficiency**
3. **Compatibility**
4. **Usability**
5. **Reliability**
6. **Security**
7. **Maintainability**
8. **Portability**

1 Functional Suitability

The IATI Core Software as defined by the Technical Stocktake will be formed by a microservice architecture with each service providing an Application Programming Interface (API) as the manner in which it provides interoperability with other services. The concept of an API Gateway has been introduced by the Technical Stocktake and would provide the comprehensive and only public-facing Web API. In addition to APIs , a number of Graphical User Interfaces (GUI) will provide functionality to human end users in addition to APIs..

We have, therefore, two distinct interface types against which to assess functional suitability:

- APIs
- GUIs.

In both instances functional completeness should be established following an appropriate requirements gathering exercise.

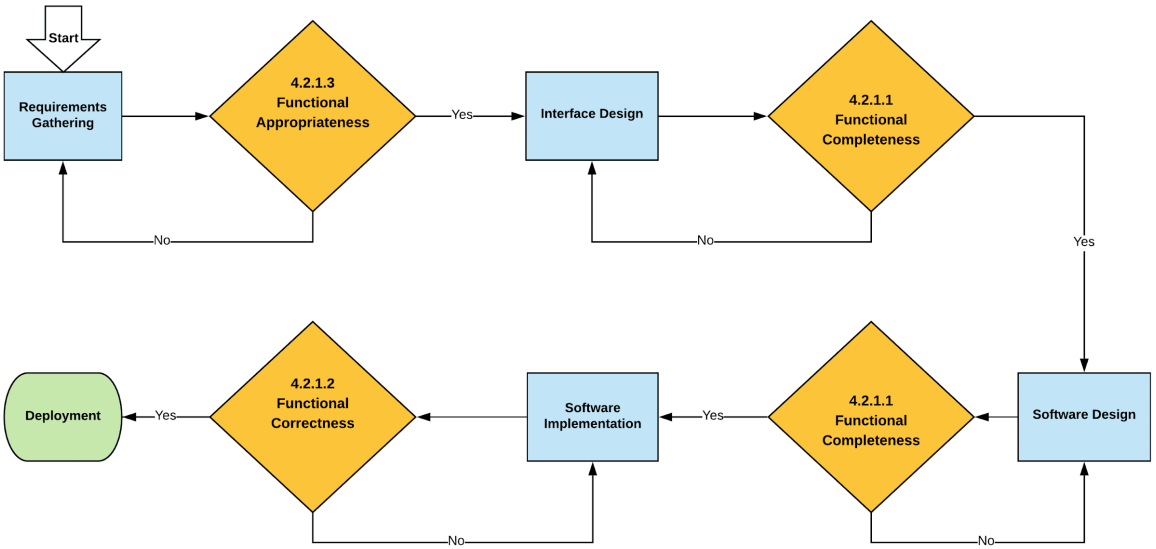
The requirements gathering phase should consider:

- For APIs, the services required of the component implementing the API by other components in the system.
- For APIs, the requirements of potential external users (grouped by those who share user stories e.g. system integration) who may have interest in integrating with the system and engaging them in discussion to elicit their requirements
- For GUIs, a cohort representing a broad range of potential users should be identified and their common requirements established.

Where APIs are concerned, we will approach their design, and the formal definition of their functionality, with a Design By Contract approach (https://en.wikipedia.org/wiki/Design_by_contract). Each API will be well defined, documented and agreed upon by relevant stakeholders before any implementation work begins.

Where GUIs are concerned we will apply a similar approach, where the contract is defined by UX design documents, such as graphic designs, wireframes, and user journeys.

The chart below shows at which stages in the development processes functional suitability should be assessed:



1.1 Functional Completeness

Functional completeness will be assessed against the following:

- For APIs, a formally prepared and accepted API Contract
- For GUIs, a formally prepared and accepted Design Document

This assessment will ensure that all functionality as defined by the API Contract and/or Design Documents is included and that no extra functionality has been included without agreement with the Technical Team..

1.2 Functional Correctness

Functional correctness should be assessed by formal testing against each aspect which comprises Functional Completeness. These tests will vary depending on the nature of function which the test; for example, GUI functionality might be tested for completeness by well defined User Acceptance Tests, an API by integration tests, a discrete unit of logic within the code, such as perhaps a currency conversion function, by unit tests.

1.3 Functional Appropriateness

Functional appropriateness will be ascertained by the Requirements Gathering exercises, which therefore must be completed before either the API Contracts or Design Documents are completed.

2 Performance Efficiency

The environment in which the IATI Core Software operates offers a set of performance challenges which, if not unique, are very different from other fields; e-Commerce, for example, or high-traffic media. We will not be numbering our users in the millions per month, nor hundreds nor even tens of thousands. However, each IATI user is of high importance, and many will be reliant on a robust and consistent performance from the core technologies.

An additional challenge is presented by the dynamic nature of the IATI publishing environment.. Whilst prior trends can be used to gain a level of insight, publishing metrics will remain unpredictable, as there are no limits in place as to how many IATI files may be published or updated in any given period of time, and nor would any limit be desirable.

2.1 Time Behaviour

Whilst the implemented interfaces should perform efficiently and as promptly as possible, robustness and consistency should have priority over nominal speed. That is to say, it is preferable to guarantee that any given service would always return its response in n milliseconds, as opposed to occasionally $n/2$ milliseconds but occasionally $2n$ milliseconds. In this manner we will foster trust in the interfaces.

Similarly, where scheduled processes are concerned, it is preferable to ensure the process should always complete when expected, as opposed to occasionally delivering more quickly or more slowly. So, if it is known that a process can reliably complete once every 24 hours, we should aim to provide a service level for that process where it *only* but *reliably* completes every 24 hours, and neither more quickly nor more slowly. Again, this will foster trust with potential users of the system.

2.2 Resource Utilisation

Whilst effort should be made to ensure that resources are utilised efficiently, in relation to the need for robustness and consistency (see 2.1) effort should also be made to ensure that no part of the infrastructure becomes over-utilised and as a result adversely affects the system.

Where appropriate, monitoring and automated notifications should be in place, and performance thresholds set appropriately to the requirements of each portion of the infrastructure.

2.3 Capacity

As the IATI environment is dynamic, all components should, within reason, be designed in such a way that might handle a surge in any given time period.

3 Compatibility

Compatibility between components is a key predicate of the microservices architecture as proposed in the IATI Technical Stocktake. Each component must provide interfaces allowing the full range of functionality required of it by all other components in the system (see 1.1).

3.1 Co-existence

Each component should operate independently of any other, and should require no other component to *adequately* respond to requests to the API it implements.

This does not mean that any given component must *successfully* return a set of results independent of all others. It may be that Component A requires an adequate response from Component B to successfully return a set of results, and does not receive that adequate response. In this instance it should report to the user of its API that it could not do so (and indicate the reason why). This would be considered an *adequate* response.

3.2 Interoperability

Interoperability between components is essential within a microservice architecture.

In the IATI Core Software, this should be achieved by designing, and subsequently implementing, *all* new and ported APIs as Web services with close adherence to the Representational State Transfer (REST) architecture - see https://en.wikipedia.org/wiki/Representational_state_transfer

4 Usability

This defines the degree to which a product or system can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use. This should be adapted per component and the definitions here used as a guide to aid development when functional requirements have been established. Usability should be assessed at the Interface Design stage.

4.1 Appropriateness Recognisability

In the case of a public API, a user with a good basic knowledge of IATI should need no further documentation other than the API documentation itself to ascertain if the services provided are appropriate for their use cases.

In the case of a GUI, a user with a good basic knowledge of IATI should need no further information other than that presented by the interface to ascertain if the services provided are appropriate for their use cases, regardless of technical competence level.

4.2 Learnability

Learnability is subsequent to **4.1 Appropriateness Recognisability.**

For a public API, a user with a good basic knowledge of IATI should need no further documentation other than the API documentation to learn how to use the interface.

For a GUI, a user with a good basic knowledge of IATI should need no further information other than that presented by the interface to learn how to use the interface.

4.3 Operability

In the case of an API, operability should adhere to, and be defined by, **3.2 Interoperability**, following a REST architecture.

In the case of a GUI, and in particular a Web GUI, modern conventions should be applied where possible to ensure that a user with a good working knowledge of modern Web interfaces will be comfortable using the GUI.

4.4 User Error Protection

In the case of an API the operability should be in line with, and defined by, 3.2 Interoperability, and adhere to a REST architecture.

In the case of a GUI, particularly a Web GUI, modern conventions should be applied to ensure that input validation happens both client and server side, that where possible the values a user can input are limited to allowed values only, and that all user errors are reported by a clear message to the user with appropriate guidance on how to rectify the error.

4.5 User Interface Aesthetics

In the case of an API the aesthetics should be in line with, and defined by, 3.2 Interoperability, and adhere to a REST architecture.

In the case of a GUI, and in particular a Web GUI, modern conventions should be applied at the interface design stage. The aesthetic design should be in the first instance a servant to the User Experience (UX) design, and in the second adhere to standard IATI branding.

4.6 Accessibility

Where possible all GUIs should adhere to the Web Content Accessibility Guidelines - Web Content Accessibility Guidelines (WCAG) 2.1

5 Reliability

Reliability is the degree to which a system or component performs its specified functions under specified conditions for a specified period of time.

5.1 Maturity

The system or component must work reliably under normal operation. Normal operation should be defined at the Requirements Gathering stage of development.

5.2 Availability

The system or component should make every effort to understand its own state of availability.

There should be processes in place to monitor for instances when a system or component has become unintentionally unavailable.

Systems should not be designed with periods of intentional unavailability or downtime. If that proves impossible, the IATI Technical Team should be consulted.

5.3 Fault Tolerance

Every effort should be made to catch and log faults in the software, and those faults should be recorded to a log file, or similar.

Faults should be categorised to at least two levels - *warning* and *fatal.*

A *warning* error should be logged, and the software should be able to continue to operate. A *fatal* error should be logged and an appropriate notification message should be sent.

5.4 Recoverability

Every effort should be made in the software design stage to ensure that in the event of an interruption or a failure a product or system can either recover, log a *warning* message and re-establish the desired state of the system, or if this proves impossible log the *fatal* error and send an appropriate notification message.

6 Security

Security concerns the degree to which a component or system protects information and data so that persons or other components or systems have the degree of data access appropriate to their types and levels of authorisation.

6.1 Confidentiality

IATI is an open standard to which open data is published. Therefore, IATI data itself has no inherent confidentiality constraints. However, in an eventuality where personally identifying data might be stored the software should adhere to the GDPR regulations (https://gdpr-info.eu/). In cases where admin level access is required - for configuration, maintenance or any other purpose - suitable authentication and access restrictions should be implemented.

All IATI Core Software source code must be released openly and licensed under the GNU Affero General Public Licence (https://www.gnu.org/licenses/agpl-3.0.en.html) unless otherwise explicitly agreed with the IATI Technical Team before implementation.

6.2 Integrity

All deployed scripts and binaries, and all data relating to the operation of the component or system, should be secured using industry standard practices to ensure their integrity and that no unauthorised alteration can take place.

6.3 Non-repudiation

All access to infrastructure should be secured and authorised access should happen only via credentials specific to the person making the access to ensure that any changes made can be traced to the individual who made them.

Similarly, any admin-level user access via a GUI should only be via credentials specific to the person making the access, and where applicable user action logs should be maintained.

6.4 Accountability

As regards APIs, it is best practice that all users should be first authenticated and identified (see 6.5) before being able to use the services that an API provides. This reveals accountability for the actions of the user, and also helps in the protection against the potential of denial of service attacks. For IATI this may not always be desirable, appropriate or practical to do; we will assess on a service-by-service basis.

6.5 Authenticity

As regards APIs, it is best practice that all users should be first authenticated and identified before being able to use the services that API provides. As above, we will assess on a service-by-service basis.

As regards GUIs, the expectation is that there be no reason to authenticate and identify general users of the interface. In the case of admin level access, appropriate authentication should be implemented.

7 Maintainability

It should be assumed that any given component in the IATI Core Software will have a long period of service. Every effort should be made to ensure the ease of maintainability of each component, considering both potential changes in the environment in which it operates and potential changes in the requirements of the functionality it offers.

In addition to the points below, the technologies used to implement each component - including but not limited to platform infrastructure, persistent datastores, programming languages, libraries and frameworks - should be agreed with the IATI Technical Team in advance of implementation, and any subsequent introduction of new technologies should also be agreed with the IATI Technical Team.

7.1 Modularity

The IATI Core Software, as scheduled by the IATI Technical Stocktake of 2020, is defined as having a microservice architecture. Therefore, each component should be discrete, have a clearly defined purpose, and not extend its functionality beyond the functional completeness required for that purpose. In addition, it should implement the API contract or contracts as designated to it to allow other components to leverage that functionality (see 3.2)

7.2 Reusability

Part of the value of a microservice architecture is the potential reuse of each microservice. APIs should be designed with such in mind (see 3.2)

7.3 Analysability

To gain abilities to assess the impact on other components or systems of an intended change to a component, API users should ideally be authenticated, and their actions should be logged (see 6.5)

To gain abilities to diagnose a product for deficiencies or causes of failures, every effort should be made to include adequate logging of faults and, where useful, state. (see 5.2, 5.3 and 5.4).

7.4 Modifiability

To ensure a component can be effectively and efficiently modified without introducing defects or degrading existing product quality, comprehensive integration tests should be written and maintained against each API service (see 7.5).

7.5 Testability

Comprehensive integration tests should be written and maintained against each API service.

Where applicable, unit tests should be written and maintained against appropriately testable units of code.

Testing of GUIs should include formal User Acceptance Tests (UAT) conducted by the IATI Technical Team.

8 Portability

Portability concerns the degree of effectiveness and efficiency with which a system, product or component can be transferred from one hardware, software or other operational or usage environment to another.

8.1 Adaptability

Using only the technologies as agreed with the IATI Technical Team before implementation will aid our ability to adapt the software effectively and efficiently for different or evolving hardware and other operational or usage environments (see 7).

8.2 Installability

Clear, logical and comprehensive installation instructions should be included alongside the application code.

8.3 Replaceability

We design our APIs by Design by Contract (see 1). All microservice components will be responsible for implementing at least one API, and their functionality will be defined by that of the API. Therefore, any microservice should be easily replaceable by re-implementing the same API or set of APIs.

Contact: For any questions or comments about IATI Non-Functional Requirements (NFRs) please contact the IATI Technical Team: support@iatistandard.org.