Chapter 9

Quality in Mashup Development

Figures
10.2 Component Quality

API quality
- Functionality
  - Interoperability
  - Compliance
  - Security
- Reliability
  - Maturity
  - Reputation
- API Usability
  - Learnability
  - Operability

Data quality
- Accuracy
- Timeliness
- Completeness
- Availability

Presentation quality
- Presentation Usability
- Accessibility

Fig. 10.1 Dimensions, attributes, and sub-characteristics for the quality of mashup components [56].
10.2 Component Quality

**Interaction overhead**
- Diffusion
- API operability

**RESTful service**
- SOAP/WSDL

**JavaScript component**
- low
- high
- narrow
- wide

**Standard structure**
- Parsing
- Data operability

**Parameter-value pairs**
- JSON
- XML
- ATOM, RSS, GData
- yes
- no

**Authentication model**
- No
- authentication
- API key
- Developer key
- User account

**Fig. 10.2** Possible security mechanisms adopted by mashup components and examples of component security scores [56].
Within the API quality dimension, usability refers to the ease of use of the API, which is different from presentation usability that we will discuss later in this section. API usability can be measured in terms of: 

- *Understandability*: Given our black box approach, understandability can be evaluated by considering the availability of the component documentation. The support offered to mashup composers by means of examples, API groups, blogs, or forums, and any other kind of documentation is particularly relevant and increases these quality attributes.

- *Learnability*: Operability also affects the ease of use of a component API. It depends on the complexity of the technologies used at the application and data layers, and of the adopted security mechanisms. Operability at the application level can be evaluated by considering the diffusion of both protocols and languages used in the API development as well as the possible interaction overhead that might derive from their complexity. The diffusion of technologies also generates the creation of a common knowledge that in a sense might facilitate learning how to use them. Similarly, the component operability is higher when the programmatic interaction with the

**Fig. 10.3** Operability of the technologies used at the application and data level [56].
Within the API quality dimension, usability refers to the ease of use of the API, which is different from presentation usability that we will discuss later in this section. API usability can be measured in terms of: understandability, learnability, and operability.

Given our black box approach, learnability and understandability can be evaluated by considering the availability of the component documentation. The support offered to mashup composers by means of examples, API groups, blogs, or forums, and any other kind of documentation is particularly relevant and increases these quality attributes.

Operability also affects the ease of use of a component API. It depends on the complexity of the technologies used at the application and data layers, and of the adopted security mechanisms. Operability at the application level can be evaluated by considering the diffusion of both protocols and languages used in the API development as well as the possible interaction overhead that might derive from their complexity. The diffusion of technologies also generates the creation of a common knowledge that in a sense might facilitate learning how to use them. Similarly, the component operability is higher when the programmatic interaction with the

**Fig. 10.4** Operability of the security mechanisms
Moving the focus to the quality of mashups, the problem is how to achieve quality combinations starting from components that themselves are characterized by their own quality properties. A specific quality model is needed because the notion of quality already investigated in Software and Web Engineering may be partly appropriate to measure the internal quality; the mashup code indeed is limited to the integration logic, while the other feature of a mashup derive from the components. Traditional quality models may then result insufficient to cover the mashup external quality. Some studies found that most quality dimensions and their decomposition into finer-grained attributes, even considering only Web application development in general, are domain-dependent. In other words, there is a strong impact by the application domain on the usefulness of quality models. Thus the component-based nature of mashups and the importance of the integration logic in shaping up the mashup behavior must not be neglected.

Our analysis on the programmableweb.com repository, for example, showed that criteria that are cornerstone for the usability of traditional Web applications, e.g., the richness of links and intra-page navigation, or the readability of text, could not be adequate for mashups. To understand this aspect, it is sufficient to observe that map-based components do not necessarily show text; rather, they visually render location markers on a map. Nevertheless, the effectiveness of such components is still high and proved successful in several contexts.

**Fig. 10.5** Dimensions, attributes and characteristics in a mashup quality model [55].
A mashup's behavior depends on the behavior of its components. Each component has its own application feature set $FS_i$, data set $DS_i$, and possibly multiple visualizations $VS_i$. To fulfill the mashup requirements, smaller portions of features (the Situational Feature Set $SFS_i$), smaller portions of data (the Situational Data Set $SDS_i$) and one or few more visualizations (the Situational Visualization Set $SVS_i$) are considered, depending on the specific needs that the mashup is supposed to satisfy. To stress the importance of having some form of integration, we assume, as minimum requirement, that for each component included in the mashup at least one operation in a component is coupled with an operation in another component, or that the data sets originating from two different components are merged and visualized through a unified visualization, or that two different visualizations have some kind of synchronized behavior.

The integration of the three sets made available by the different components generates in the mashup the Real Data Set $RDS$, the Real Feature Set $RFS$, and the Real Visualization Set $RVS$, which correspond to what the mashup actually offers. In the best case, the real sets should equal the corresponding ideal sets; however, it is possible to observe how in many cases some elements of the ideal sets are not covered by the real sets. Assessing the quality of the mashup integration strategy thus consists of assessing the quality of such real sets, and in particular in comparing the real sets made available by the composite application with the ideal sets at the origin of the mashup conception (this is useful to evaluate whether the mashup goals are satisfied), and with the component situational sets, to understand whether the integration introduces new capabilities, thus evident advantages, with respect to the independent use of the single components.

**Fig. 10.6** Data sets involved in information integration in mashups [57].
We will see next in this section how a sensible definition of component roles has an evident impact on the integration logic quality, and how, in the assessment of mashup quality, the identification of roles can suggest how to interpret and weigh the quality of single components within aggregated measures.

### 10.3.3 Composition quality

Composition quality aims at evaluating the suitability of the integration strategies adopted in a mashup, and in particular of the chosen components and the composition patterns with respect to the ideal features, data and visualization originally conceived for the mashup. It can be decomposed into aggregated quality, added value, component suitability, consistency and availability.

**Aggregated quality** refers to the impact that the quality of single components has on the overall quality of the composite application, and how it can be tuned based on the role that the components have in the composition. Its estimation is achieved by aggregating the quality measures of the different components, for example computed on the basis of the component quality model that we analyzed in the first part of this chapter. However, it is also important to weigh such measures to reflect the role of the components in the mashup. Based on our analysis of component roles and composition patterns, it emerges that master components, being central points of synchronization, have in general a major influence on the mashup quality – remember that a

---

**Fig. 10.7** Examples of (a) the slave-slave pattern, (b) the master-slave pattern, and (c) the master-master pattern as observed in three online mashups [57].