How-to: Functional documentation in Enterprise Architect

Carolien de Langen
Chan Kerste
©2019
Inhoud

Scope of this white paper .................................................................................................................................................... 2
  In scope .............................................................................................................................................................................. 2
  Out-of-scope ..................................................................................................................................................................... 2
The case: RunTrack app ......................................................................................................................................................... 3
Where to start? ....................................................................................................................................................................... 3
Document the Screens ........................................................................................................................................................... 3
  The navigation map ............................................................................................................................................................ 4
  A detailed screen design ....................................................................................................................................................... 5
  Views ................................................................................................................................................................................... 7
Document the Use cases .......................................................................................................................................................... 9
  The use case model ............................................................................................................................................................ 9
  The use case specification .................................................................................................................................................. 10
Document the System interface .............................................................................................................................................. 14
  1. Service component with interface connector .................................................................................................................. 17
  2. Data selection Activity element ......................................................................................................................................... 18
  3. Request and Response classes .......................................................................................................................................... 19
  4. Data model classes ........................................................................................................................................................... 21
  5. Transformation Actions ...................................................................................................................................................... 22
    Components with their interfaces ....................................................................................................................................... 23
Document the Data model ....................................................................................................................................................... 24
About us .................................................................................................................................................................................. 25
It is time to take the next step in making your functional documentation. Your company bought Enterprise Architect, but you find it hard to determine where to start. It is even more of a puzzle to make the different diagrams in the functional documentation traceable. The last couple of years we found out that a lot of our customers are struggling with this question. In this white paper we explain how you solve this puzzle in Enterprise Architect and have traceable functional documentation.

**Scope of this white paper**

We scoped this white paper so the information is good to understand.

**In scope**

In this white paper we explain how you can:

- Document the Screens
- Document the Use Cases
- Document the System interface
- Document the Data model

**Out-of-scope**

In this white paper we will not explain how you can:

- Document Requirements
- Document the Business Rules*
- Set up the project in Enterprise Architect
- Document the Business Object Model
- Document the Class Model

* We don’t explain how to use the elements that Enterprise Architect provides for documenting Business Rules. But we do explain how you can describe logic in the detailed screen design, use cases and system interface that descents from the business rules.
The case: RunTrack app

Because we both like to run, we decided to write a case about tracking our runs. Almost everybody who runs uses a run tracker nowadays. This is a short description of the case:

*To track the training, we are using the app RunTrack. In RunTrack a new run can be tracked by clicking on ‘Start’. The app uses GPS data of the phone to track distance. During the training the ‘Pause’ function stops tracking until ‘Start’ enables tracking again. When the training is finished, ‘Finish’ will stop tracking.*

*Now RunTrack has measured the distance and time period for a training. In ‘My Trainings’ the results are presented per training. A graph shows the progress of all trainings together.*

*Runners that prefer to run with a watch to track their training can import the watch results in the RunTrack app.*

Where to start?

To document the functional design in Enterprise Architect there are a couple of starting points. Register the requirements, make the use case model, create a business object model or maybe start with the screens. It depends on your own preference where to start.

When real users are involved, we normally start eliciting the requirements from the users. Based on the requirements gathered, we pick the preferred starting point. Because requirements are out-of-scope for this white paper we started with defining a couple of use cases. Now we had a global idea about the RunTrack app. Because this is an app and the front-end is important to the end users, the next step was to make the screen sketches.

Document the Screens

Documenting screens in Enterprise Architect is rather straightforward. However, there are a couple of toolboxes in Enterprise Architect to choose from. We chose the ‘User Interface’ toolbox because it is easy to use. We found out (in Enterprise Architect version 13) that the more graphical nice-looking Win32 toolbox screens had some lay-out problems: when adding a new element to the screen, Enterprise Architect reshuffled all other elements on the diagram.
The navigation map

We started to make the Navigation Map of the RunTrack app. This gives a good overview of all screens in the app and how they are connected.

The diagram is created by adding ‘Screen’ elements to the diagram. To connect these elements, we used the connector with an arrow. The arrow points to the Screen element the user navigates to. It is possible to navigate both ways in one connector, but we chose not to put this in the diagram. We made this decision because we also put the action that triggered the navigation in the connector. If both actions are placed in this one arrow, then a mistake is quickly made. Of course, we show all navigation options in the detailed screen design.

In the example navigation map on the next page, you can see that the user can navigate from the screen ‘Login’ towards the screen ‘My trainings’ with the OK-button. Another choice we made is not to put the OK-button in the diagram. Otherwise the diagram has too much information to be understandable.

Another choice we made in the navigation map is to make a separate screen for a new training and an ongoing training. An ongoing training has far more information in the screen than a new training. It is much easier to explain what happens on the screen when they are described separately. E.g. if a new and ongoing training were described in one screen, we had to add a lot of constraints to the screen. The screen wouldn’t be readable for the reader of the functional design.

With a Decision element in the navigation map it is clear that the Record-button navigates the user to a new training or an ongoing training.
A detailed screen design

The detailed screen design is a guide for the developer to build the screen in de app. If the developer follows the design, then the screen is built as the user has requested. For the RunTrack app we made the following detailed screen design, which contains this information:

1. Screen elements
2. Data source
3. Navigation to other screens
4. Related use cases

1. Screen elements

In the detailed screen design the screen is put in a diagram. On the screen we added labels, text fields and we used panels to group the data on the screen. Each element on the screen is described in detail in the Note field (under element properties). This gives the reader an understanding of what kind of information the element provides to the user. Under element properties you can also describe the logic that might be applied.
2. **Data source**
Other important information for the developer is where to get element data. E.g. where the total distance of the run is provided: in the cache of the app or via a system interface like a REST service. This is discussed in 'Document the System interface'.

3. **Navigation to other screens**
The detailed description of the elements is not the only information in the detailed design. In the diagram the navigation to other screens is also described. This gives the reader the information from which screen this screen can be called and which screen the user can open from the screen.

4. **Related use cases**
In the diagram also a link is added to the use case(s). This gives the reader an idea in which task(s) the screen is used.
Views

When making the screens we found out that sometimes all this information in the detailed design is a little bit too much. Especially screens with a lot of data on the screen. To make the detailed design understandable again, a good solution is to make a 'layout' view and 'implementation' view of the screen. Below we show an example of the layout and implementation view of the screen ‘Run details’.

On the left-hand side everything is described in one diagram. The screen details, the navigation, the use case and the system interface. On the right-hand side are the elements represented in the right place on the screen in the 'layout' view. Also the navigation and the use cases are in this view. This view is good to show to users. Also displayed on the right-hand side is the 'implementation' view. In the 'implementation' view the reader can see where to get the element data. See ‘Document the System interface’ for a more detailed description of documenting the data needed for the app. In this view e.g. labels can be left out, which makes the diagram readable for developers.
A quick recap of the choices we made to make the screens understandable for the readers:

- In the navigation map a connector navigates only one way.
- Buttons or links that trigger the navigation are not in the navigation map to keep it clean. In the detailed design of a screen this information is described.

For documenting screens in detail, we have a couple of tips:

- If an element description is important to display in the diagram, add a Note element to the diagram and select element note in the connector to display the element description in the diagram. See diagram below to see how you can connect the note element for the field ‘Current speed’.
  In fact, we advise to make all notes visible on the diagram so that the useful information is not hidden in the properties of the element.

- When designing the screen always keep in mind that the design must be understandable for the reader.
- If there is a lot of data on the screen, split the detailed screen design up in a layout and implementation view.
Document the Use cases

Screens are important in the functional design, but the reader wants to know which task he can accomplish with the given software, without browsing through all the screens. For the RunTrack app the reader wants to ‘track his sunny Sunday morning run’. These tasks are described in the use cases.

The use case model

Making the use case model is a simple way to get an overview of the functionality of the RunTrack app. Basically it is one sentence for each use case. This sentence is divided in the person or system who is executing the task and the task on hand. In other words, the actor is the person or the system and the use case is the task to execute. This gives a quick overview what you can do with the app and who can do it.

In the diagram below the use case model of the RunTrack app is displayed.
The use case specification

Documenting a use case model in Enterprise Architect is done quickly, describing each use case in detail can be more challenging. But we put all this effort in the detailed design so we know that the task executed by the user is according to the requirements, the developer knows what he has to build and the tester knows how to test the use case thoroughly. And at the end the software is going live without production problems that could have been prevented.

For each use case in the use case model a detailed design (also called ‘use case specification’) is made. We started with creating the happy scenario (or basic flow) of the use case. After that we added alternate scenarios and the logic that describes when which scenario is applied.

Structured scenario
The flows are described in a ‘Structured scenario’. The ‘Structured scenario’ can be found in a tab under ‘Scenario’ in the Element properties. In the diagram below an example of the structured scenario of the use case ‘Track run’ is presented.

We use the structured scenario instead of the ‘Textual Description (the other tab under ‘Scenario’) for a couple of reasons. The structured scenario forces you to describe each step in a specific format. This is very handy because every information analyst must use the same format which improves the readability of the functional design.
**Activity diagram**
If you prefer an activity diagram in your use case specification, then the structured scenario has another huge advantage over the textual description. From the structured scenario it is possible to generate the activity diagram in Enterprise Architect. This is a handy feature and is improved over the years. Especially testers are fond of the activity diagram to create the test scenarios.

There are a couple of things worth to mention when creating the scenarios in combination with generating the activity diagram. The first time Enterprise Architect generates the activity diagram the lay-out isn’t spot on. E.g. there is a lot of space between the elements. We tend to improve the generated diagram so the readers can only focus on the information instead of making remarks about the drawing techniques used by the information analyst. However, we don’t want to lose this extra work the next time we are generating the diagram after some changes have been made.

So the next time the activity diagram must be generated, choose the option ‘Synchronize elements in existing’ and check ‘Preserve diagram layout’ instead of ‘Overwrite existing’. Enterprise Architect will only add the changes and leave everything else as is. In the diagram below this option is displayed.
This option is very important to us, because in the generated activity diagram we like to add extra information like the screens. We add the screens because the reader then can see which screens are used in the executed tasks. An example of an improved generated activity diagram is the following diagram.

In the above Activity Diagram, the reader can also double-click on the screen to see the detailed screen design. This is the power of putting the functional design in Enterprise Architect (traceability) rather than documenting each screen and use case in e.g. Microsoft Word document.
Traceability: composite diagram

The traceability option in Enterprise Architect is the option we use the most. The reader can double-click on every element in our RunTrack app with the ‘glasses’ represented in the diagram. Getting these glasses on the element is very simple. Select the element in the diagram with a right mouse-click, choose under ‘New Child Diagram’ the option ‘Select Composite Diagram’. For the proper mouse-clicks see the following screen print.

Select the desired diagram and traceability is made. Every element can be traced to another diagram.

A quick recap of the choices we made:

- To design the use case specification, we used the ‘structured scenario’ for describing the flows instead of the ‘textual description’. This way the activity diagram can be generated and all information analysts are using the same syntax for describing the scenarios.
- We improved the generated activity diagram layout-wise and added screens and system interfaces that will be used. In one glance the reader knows where the data is stored and which screen is executed in the task.

For documenting use cases we also have a couple of tips:

- Add ‘composite diagram’ hyperlinks to use cases, screens and system interfaces to make the functional design traceable between diagrams.
- Always take a step back and consider if the diagram is understandable for the reader.
Document the System interface

It is obvious why we document the screens and use cases. But the screens and use cases aren’t enough to functionally describe software as a whole. People need answer to the following questions: “Where does this piece of data come from?” and “Which part of the system did alterations on the data before presenting it on a screen?”.

We can answer these questions by modelling the system interface in Enterprise Architect. With this approach our readers are able to track an element that is displayed on the screen back to its source. Also, it is possible to see any alterations being made by the system(s) in between.

We also discovered that it also works the other way around: “Where do we use this piece of data anywhere else in the system?”. By using the modelling technique we describe, our readers could even get an answer to this question.

In the system interface we document every request the software makes and the answer the software receives. The request can be: ‘Give me all the tracked runs’ with the answer: ‘All the tracked runs with detailed information’ but a request can also be: ‘Store the tracked run’ with the answer: ‘The tracked run is stored successfully’.

These requests are called the request messages and the answers are called the response messages of the system interface. In the response message the answer is given in sending information in response attributes. By connecting the response attributes to screen elements, we managed to show where the attributes are used in the system. Or, what attribute is used to see the value on the screen.
See the diagram below for an example how the answer of the request ‘GET training/(trainingid)’ is mapped to the screen ‘Training details’.

Because documenting a system interface can be a hard job and sometimes a bit technical, we explain the system interface of the request ‘Get training’ in detail. In the example we modelled, we used an interface with REST services. But the model could also have been used for XML messages or CSV-files.

First of all, these are the basic elements in the diagram of a system interface (the diagram on the next page shows the elements):

1. **Service Component with interface connector**: to see what component in the system landscape the interface belongs to. See also ‘Service components with their interfaces’.

2. **Data selection Activity element**: describes the selection steps within the interface with logic in a flow of actions. It has a request and response ‘Activity Parameter’ to connect the Request and Response classes.

3. **Request and Response classes**: to link the data model to the request message or response message.

4. **Data model classes**: the source of the system interface message. With these you can get, but also store data.
5. **Transformation Actions**: any transformation logic on the data is described in actions. For complex transformations you can include an Activity diagram. Now you can see easily what transformations are done by the interface itself!

In the diagram on the next page the basic elements are shown.
1. Service component with interface connector

Interfaces are placed upon a Component. They cannot exist without it. In our case we have a Service component named RunTrack. The first thing to do is to put the Service component on the diagram by dragging it from the Project Browser (See step ‘1. Drag Service component on the diagram’ in the diagram below).

When this is done, you can drag the Interface onto the Service component. It appears as a lollipop stick (see step ‘2. Drag interface element on the Service component’).

Now it’s time to connect the interface with the request ‘Get training’ (See paragraph ‘2. Data selection Activity element’ in detail). The request is represented as an ‘Activity element’ in the diagram. You should connect the Activity element with the Interface via a Dependency relationship.
2. Data selection Activity element

A system interface does not only consist of input and output. There is also logic to get the right answer. More technical: retrieve the right data. Which data set is retrieved? Everything? Only for a given ID as given in the request? Or only those items with a certain status? The answer to these questions is part of the activity element.

With a flow of actions, you can describe the logic that is performed to get the set of data. See diagram on the next page. You don’t describe transformations on attribute level here. This is done with ‘5. Transformation actions’.

The example in the diagram only has one action step. If the logic has too many steps to fit in your diagram, you should use an Activity with composite diagram so that by double-clicking on the Activity you will get to the extended activity flow diagram that you need. See ‘Use case specification’ how to make the composite diagram.

If you use attributes to set a filter on the data, then use the same attribute names as used in the system interface. This will make the short description more like pseudo code and makes it exact.
3. Request and Response classes

The information that is requested and the information that is answered are described in the request and response classes. Each request or response will have two Enterprise Architect class types: one Interface class and one or more regular classes.

The Interface classes have one function only: to be able to connect the Request and Response classes to the Activity parameters. You can just name the Interface classes the same as the system interface name and use Stereotypes ‘Request’ and ‘Response’, see image below.

The regular classes have more detail. These are modelled in the structure of your system interface message (request and response messages). Add classes and attributes with the same names and types as in the actual message. For a regular class the Stereotype is empty.
Now link Interface class to the top regular Request or Response class. See image below.

Once the structure of the request and response message is known, the last step is to document the source of the messages, see ‘4. Data model classes’. 
4. Data model classes

The Data model classes are the source of your system interface message. With these you can get, but also store data.

Drag the classes that you use for the system interface from the Data model (see 'Document the Data model') in the Project Browser into your diagram.

Now you can link the Data model attributes to the Request and Response class attributes. You do this by making a Dependency relation between the classes. When you go to the properties of the Dependency relation, you can fill out the Source and Target attributes by selecting them from a list of attributes. See the image on the next page.

If data is not one-on-one to be retrieved or stored you can put an Action element between the data model source attribute and the Request or Response attribute. This describes the transformation logic that needs to be done by the system interface. See ‘5. Transformation Actions’.
5. Transformation Actions

A Transformation Action is an Action element that is put between one or more Data model attributes and the linked Request and Response class attributes. Use the exact attribute names in your description to precisely describe the logic that needs to be applied.

If the transformation logic is complex, you can use an Activity element with a composite diagram to describe the complex transformation.

Now, with one glance at the diagram, you can see what logic is applied to the data source before it is provided or stored by the system interface. This helps in trying to discover any bug that might appear.
Components with their interfaces

Because software tends to have more than one system interface, it is good to have a diagram with all interfaces together. This diagram is called the Component Interface diagram. It gives a clear overview on the source system(s) your application uses.

On the Component Interface diagram you can add the Component to represent the providing system and add Interfaces (lollipop sticks) to this component to represent the system interfaces. You also need to have the component with lollipop sticks as part of your model in order to link to it in other diagrams, so you already create the diagram while adding interfaces.

To be complete you can add the Activity elements of the interfaces as well. These elements are connected to the Interfaces with a Dependency relationship. This way the Component Interface diagram is also a fast way to be able to browse to any system interface diagram by double-clicking on the corresponding Activity element.
Document the Data model

The data model contains the classes and attributes of the source for the RunTrack app. In our example we have only one data source, but you could have an application where the system interface retrieves data from multiple data sources. In that case each data source will have its own data model diagram.

In the data model diagram we register the following:

1. Classes
2. Relation between classes
   a. Multiplicity (e.g. 0..1)
3. Attributes
   a. Name: use the same name as in the data source to make a good reference.
   b. Type: you can keep it simple (like ‘string’) or be precise (like string40).
   c. Scope: we always use ‘Public’.

The classes with attributes are used in the diagram. This way you can trace for a class or attribute in which system interfaces they are used.
About us

You may be wondering who we are. We are Carolien de Langen and Chandrakant (Chan) Kerste from the Netherlands and both freelance information analyst. Since 2001 Carolien has worked as an information analyst at more than ten different companies successfully. Carolien has used Enterprise Architect since 2015. Chan started as a developer back in 2001 and since 2003 he has worked as an information analyst for also more than ten different companies successfully. He has started to use Enterprise Architect in 2006.

In 2017 we met each other as we were hired by the same company. We worked together in a Digital Transformation Program for more than a year. One of our tasks was to functionally design both our Apps in Enterprise Architect. While documenting we found out that our ideas of documenting in Enterprise Architect were alike. By sharing our ideas both our functional designs improved even more. We then came up with the idea to share our knowledge in this white paper.

*We hope this white paper gives you new ideas to create functional documentation in Enterprise Architect. Ideas that also let you see the added value of a tool like Enterprise Architect. If you have tips to improve the functional documentation further, have questions about this white paper or have questions about integration of other kind of methods you use as an analyst/designer in Enterprise Architect (e.g. Requirements), don’t hesitate to contact us or leave a comment behind.*

Contact

Carolien de Langen: carolien@beaujour.net
Chan Kerste: chankerste@gmail.com

© 2019 Carolien de Langen and Chan Kerste – All rights reserved