Date	16 NOVEMBER 2022
Team ID	PNT2022MID17378
Project Name	Job/ Skill Recommender Application

Job/ Skill Recommender Application

Up until this point we have considered "function" to refer to the "intended function" of the object being designed. Given the definition of a feature given above we have a choice of what a functional feature might be. It could be a feature that is of type function (i.e., referring to the F portion of the SBF description). It might also be a feature of any type that impacts intended use: e.g., a behavior that prevents the design from functioning. However, this seems less satisfactory.

In an interesting way, *every* feature is functional. By definition a feature is of interest because it has an effect on a process, and is detected by its effect on a goal. That process is forming an "environment" for that feature, and is interacting with it. This is consistent with the definition of a function of an object as the effect it has on its environment: i.e., by definition, a feature of the designed object is functioning in the environment of each process where that feature affects a goal.

This is not the "intended function" of the designed object, except in the case that the process is "intended use". However, a good designer who is viewing the object from a number of points of view corresponding to the phases of the life-cycle, will ensure that the designed object functions as intended when involved with all the corresponding processes.

For example, it should be easy to pack, easy to maintain, and easy to disassemble. Paying attention to a variety of dynamically detected and selected features during its design make this possible: i.e., it will function well for those processes. In fact, with respect to functional features, there is a family of types of functions, F_{pj}, where p_j might be any of the processes already mentioned: i.e., intended use, being described, assembling, packing, transporting, diagnosis, simulation, recycling, manufacturing, designing, maintenance, etc. Clearly, when viewed in this way, all features are functional.

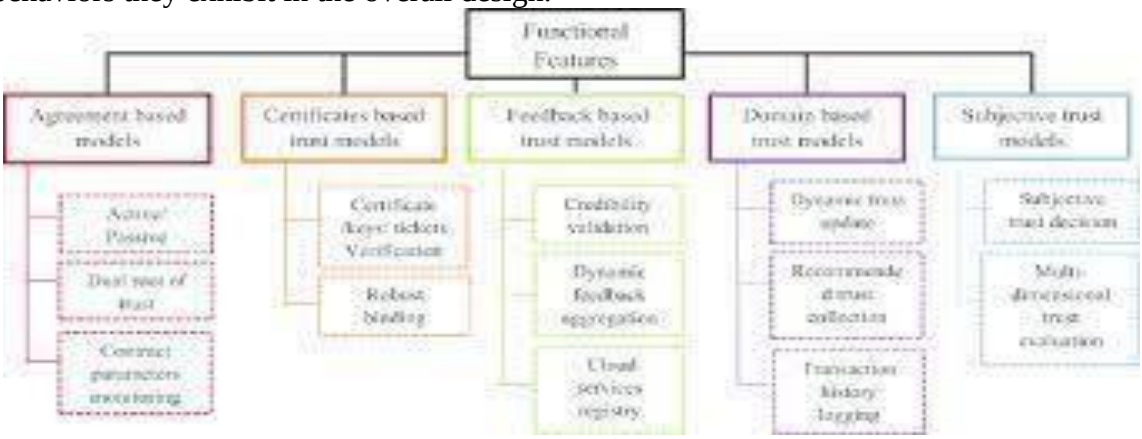
Functional Features

In addition to the types of features mentioned above, another type has been identified—Functional features: concerning function, purpose or behaviors.

Functional features have been increasingly mentioned in the literature. This has happened as CAD has been influenced by Concurrent Engineering, by theories about the design process, and has moved away from being purely geometric.

McGinnis & Ullman [1992] write that:

"Functional features include both the purpose of the design object such as support, stability, or strength and the behavior that the design object performs like lifting, gripping, or rotating. The form features embody the physical characteristics of design objects in a design while the functional features explain what purpose the design objects achieve individually and what behaviors they exhibit in the overall design."



Functional and non-functional requirements

Functional requirements

Functional requirements are features that are built to serve the products' users. They are pieces of functionality that solve particular problems for users. An example of a functional requirement would be: "User should be able to import contacts into their mail application."

Non-functional requirements

Non-functional requirements concern the operation of the system, such as technical requirements or other non-user-facing functionality. An example of a non-functional requirement would be: "System should support 100 users simultaneously."

```
>>> def func():
...     print("I am function func()!")
...

>>> func() I am
function func()!

>>> another_name = func
>>> another_name() I
am function func()!
>>> def func():
...     print("I am function func()!")
...

>>> print("cat", func, 42)
```

```
cat <function func at 0x7f81b4d29bf8> 42

>>> objects = ["cat", func, 42]
>>> objects[1]
<function func at 0x7f81b4d29bf8>
>>> objects[1]()
I am function func()!
>>> d = {"cat": 1, func: 2, 42: 3}
>>> d[func]
2
>>> def outer():
...     def inner():
...             print("I am function inner()!")
...
...     # Function outer() returns function
inner() ...     return inner ...

>>> function = outer()
>>> function
<function outer.<locals>.inner at 0x7f18bc85faf0>
>>> function() I am
function inner()!

>>> outer()()
I am function inner()!
```

Functional requirements and user personas

When creating functional requirements, it is important to understand which user persona these requirements will benefit. Realizing that there may be different user personas for your product, naming the user persona will help provide a context for your engineers.
Note: You usually do not create requirements (or build) to fit buyer personas because, as part of their buying criteria, buyers want you to build features that satisfy users.

Identify the problem your product solves

Identifying and explaining the problem you are solving for the user might be the most important part of a requirement. This enables the engineers to understand why they are building a particular feature, and it allows them to use the root problem to identify alternative solutions. To ensure that you are truly solving the user's problem, implement the Toyota method of repeatedly asking yourself (up to five times) **why** you are building that particular feature. It will help you to discover the real pain you are trying to solve and make your requirement more salient.

Example of the "why" method

> Mary (user persona) misses the bus most days going to work.
> Why?

Mary sleeps through her alarm.
Why?
Mary's alarm is not loud enough.
Why?
Mary is hard of hearing.

Defining "what," not "how" when developing a product

There are many ways to solve your user's problem. Do not tell your engineers **how** to solve a problem; only tell them **what** the solution should achieve. This will enable your engineers to develop ingenious solutions and be more motivated to find the right solution. Telling an engineer how to do his or her job is a definite de-motivator.

Using the example above, one's first instinct may be to give the engineers a requirement to build a louder alarm clock. However, this would violate the "what, not how" guideline. Instead, by stating the problem and what you'd like the solution to achieve, you have allowed for flexibility in the solution.

Improved requirement: "Mary is hard of hearing. We need an alarm clock that will allow her to understand it is time to wake up."

In this particular example, the engineers may choose to make a sound-based alarm clock, or they may decide on a totally different route, such as a light-based notification system. By outlining what the solution needs to achieve, it allows the engineers to think creatively to solve the problem.

Writing user stories to help create product requirements

Without being prescriptive, write user stories to outline the scenarios that the user must be able to achieve. For more effective results, provide your product testers with the user stories and user personas. They can use these stories to ensure that the delivered solution meets the user's needs. Ask the following questions when creating user scenarios:

What is the user doing to cause the problem?
What is the user trying to achieve?

Product requirements typically involve three types of user stories that cover particular scenarios

Daily-use scenarios: These scenarios involve the user's most frequent actions with the product (limited to two or three possibilities). *Example:* Mary turns off her alarm clock.

Necessary-use scenarios: These scenarios involve the types of actions with the product that are necessary, but not performed often (for example, configuring the system, creating a report). As these do not occur as frequently as daily use scenarios, they can take more steps to execute than those used more often.*Example:*Mary sets her alarm clock for a change in schedule.

Edge-case scenarios: These scenarios rarely occur in real-life situations, but are of some concern for engineers. It might be difficult to create these types of situations, as virtually all users will never encounter them. *Example:* Mary decides to change the alarm settings and the power goes out.