Here is a summary feedback for the recent submission by Dr. Kay that, in his words, is “an attempt to consolidate a proposal for variadic functions that combines all the ideas and requirements that have been expressed”.

This feedback is grouped into 3 sections: the good, the not so good and lastly, things that are really unacceptable.

The good

There are a number of good things in the document:

- Good distinction between parameters (in the function declaration) and arguments (in the function call)
- Clear, one-sentence definition of the term “variadic functions”
- Keyword arguments
- Using keyword arguments even in the case when calling non-variadic functions.
- Good description of “bounded variadic functions” with the exception of mentioning “virtual fixed-arity functions” (see below how this is caused by the bad and unacceptable things in the proposal).
- Good description of “array-variadic functions”
- Almost good description of “map-variadic functions” (the thing that shouldn’t be there is adding too-much implementation details how a map will be used to hold all keyword arguments)

The not so good

The obvious things that are not good in the proposal:

- The arity of a variadic function is undefined and it cannot have overloads – only a single function with that name can exist. There is no obvious, specific reason for such design decision, the practice of other programming languages shows that functions allowing calls with keyword arguments, can have additional overloads. See for example Jon Skeet’s “C# in Depth” / Overloading (section: Optional parameters (https://csharpindepth.com/articles/Overloading)). Here one can see at least two ways of disambiguating seemingly ambiguous method-calls – disambiguation by explicitly specifying the optional parameters, and disambiguation by argument names.
- The function reference of a variadic function. Obviously `someFunName#*` tells us nothing about the possible number of arguments, and the value of the arity is passed to other functions as nothing (the empty sequence `()`) , even though in all three types of proposed variadic functions, the exact number of positional argument is known, and in the case of a “bounded variadic function " the maximum number of optional arguments in a function-call is also exactly known
- “References to virtual fixed-arity functions that map directly to the variadic function”. The author had to do this in order to overcome the two bad things above – thus doing a 3rd bad thing…

- In “map-variadic” functions there is no possibility for static type checking. This provides for unlimited space of runtime “confusion-errors” where the free-text English language description is understood differently by the author of the function and by its users. Natural language ambiguity is best avoided when doing serious function design work.

The unacceptable

Some of the bad things are so bad that they are unacceptable:

- It is unacceptable to define the arity of a function as nothing (the empty sequence () or *) when the reader clearly sees that the function has M positional parameters and N optional parameters. People who believe what they see will be confused and upset.
- It is unacceptable to forbid a variadic function to have other overloads, contrary to the practice in other programming languages. If this can be done in C#, why shouldn’t it be done here?
- It is unacceptable to introduce vague, confusing terms such as “virtual fixed-arity functions“ just in attempt to fix the holes left by the document in its current form.
- It is unacceptable to prevent any static type-checking in the case of “map-variadic” functions. An obvious improvement would be “record-variadic functions”.
- It is unacceptable not to provide any disambiguation mechanisms when there are overloads that may be ambiguous if allowed. Based on existing programming languages, there are at least two obvious ways of disambiguation: by arity and by argument name.