UML diagrams difficult to read
There are too many details and/or crossing lines to see anything significant. Since the diagrams are generated, this is often difficult to avoid.

Work-arounds:
• Try different generators.
• Manually improve the generated diagram.
• Manually draw the diagram.
• Provide two diagrams: a detailed one and an overview.

No defined quality goals
The presentation lacks defined quality goals and criteria. Which qualities were you aiming at or looking for and why? Without explicit quality goals or criteria, it becomes difficult to decide what to measure.

Work-arounds:
• Write a small “method” section explaining what you did, why you did it and how.

Lacking rationale for choice of measures
There is no rationale for measuring something. Were measures chosen just randomly? Typical is some kind of “fishing”: A tool is selected in the first place and all the rest depends on the capabilities of the tool. Very often not even the reason for using a particular tool is given. “We have chosen tool X and got the following results ...”.

Work-arounds:
• See above; both “problems” are tightly connected.
• Explain how the measures are related to the “qualities” you are interested in.

Please note that it is fully acceptable to do some initial “fishing” and then define more precise quality goals and a strategy for the improvement. That means you first make a general check to identify the main problems and then set up a more detailed plan for the rest. In a real project with several 100kLOC, it is not feasible to measure everything and decide on a one by one basis what to improve.
No explanation of value ranges for measures
Measurements are presented as absolute values. The reader is not informed about the range of the value and what can be considered acceptable values. If indications are given, they are usually “fuzzy”.

Work-arounds:
- For each measure, state its range. Cyclomatic complexity and many other measures can be said to “count” things, i.e. they are Integers ≥ 1. LCOM can have very different ranges depending on the LCOM version (or tool) you are using. Henderson and Sellers for example have normalized their LCOM version to give values in the range [0..1].
- For each measure give an as precise as possible indication of acceptable ranges, if possible with a reference.

Presentation of results lacks a good overview (“översiklig”)
Measurements are often presented in text form without giving the reader a feeling for the overall picture. When tables are used, which is a good thing, most often only a single value pair element/value are shown. This does not give a good view of the distribution of measures. If a class for example has an average cyclomatic complexity of 7.5, it might still have many methods with much higher values for cyclomatic complexity.

Work-arounds:
- When presenting average/mean values one can add variance and/or min/max values to give the reader a feeling for the distribution.
- Tables are generally good to present measurement data.
- In large tables, mark “interesting” values.
- Scatter plots or bar charts can give a more detailed view of values.

Proposed changes are too generic
Proposed improvements are described in generic terms, like that a method with a particular high cyclomatic complexity or a class with low cohesion should be split up. This is, however, not always possible. A method, for example, might need to do complex stuff in a certain way.

Work-arounds:
- Look into the code in more detail and give a specific example for an improvement.

Changes are considered locally, one problem after another
Improvements fix local problems only, like splitting up a method to decrease cyclomatic complexity. More complex refactorings are not considered.

Work-arounds:
- Consider more complex refactorings that could solve many problems at once, like using certain design patterns. Using design patterns can improve coupling and cohesion in many cases.
Side effects of changes are not taken into account

This is closely related to the problem above. Some improvements have positive local effects, but might have negative effects elsewhere. Splitting up a class with high (efferent) coupling will for example decrease (efferent) coupling for all of the resulting new classes. However, if this class was quite cohesive, this might lead to a lot of coupling between the new classes.

Work-arounds:

• Look into the code to check if the proposed change has negative side effects.