Abuse Case Guidelines

1. Create use cases for all functional and non-functional requirements. Let the use cases handle all that systems stuff.

2. Assume use cases are the only requirements model you'll need.

3. Be ambiguous about system boundary. There will be scope creep anyway.

4. Name use cases as obtusely as you can. Use vague verbs, like 'do' or 'process'. Stump the reader about the goal of the use case.

5. Don't document other non-primary actors. They'll come up during coding.

6. Don't define the success outcomes of the use case. It should be obvious.

7. Don't both defining fail outcomes. You and your staff are good developers, and the end users should know how to handle any problems that might arise.

8. Don't involve subject matter experts in creating, reviewing and verifying use cases. They'll only raise more questions.

9. Write use cases as psuedocode. You haven't had a chance to use those skills in a while.

10. Write use cases in technical language that computer geeks would dig.

11. Create a separate use case for each CRUD activity. Data people will appreciate it.

12. When visually modeling use cases, use lots of extensions, generalizations, extension points and includes. They are part of the UML, aren't they?

13. When visually modeling use cases, show interactions between actors. It's interesting to see who talks to whom.

14. Don't iterative when you create use cases. Instead, write the first draft in excruciating detail.

15. Make your use cases very low-level; don't think of them as business transactions but think of them as molecular level cells in the microcosm of the business process.

16. Decompose use cases with lots and lots of extensions and includes, making lots and lots of little-bitty use cases.

17. Be sure the use case is written for designers and developers - not direct end users. Have each use case be a logical unit of work that they can code directly from.

18. Don't verify use cases with scenarios. It will cause you to revise the use case.

19. Don't use a template for use cases. Write each one with a different style and format and be sure to have different projects create their own template.

20. Repeat common procedures in multiple use cases instead of creating a separate reusable use case ("includes"). This will cause the reader to jump around the use cases to find try to find any differences, which will increase their keyboard and mouse skills.

21. Use actual people's names or job titles for actor names. We want to know who is actually going to do this stuff.

22. Create very detailed screen shots in your development tool for each use case. You'd rather skip requirements and go right to GUI design, anyway.

23. Don't worry about the testability of use cases. In fact, don't' even show testers the use cases until the end of the design phase.

24. Don't capture data needed for the user-system interaction. We don't need to worry about those "entities that don't know how to behave".

25. Forget about business rules. Even though they come up as you elicit and analyze use cases, you'll *pretty much* remember some of them when you create the Software Requirements Specification. Plus, the users don't really know their business rules anyway! So don't bother getting into all those uncomfortable issues with business rules.