Propositions

Stellingen

BY Ariadi Nugroho, Author of

The Effects of UML Modeling on the Quality of Software

1. An open question in software modeling is whether modeling delivers quantifiable benefits for software projects. As long as this question remains unanswered, it would be difficult to motivate and justify modeling activities in real software projects.
2. UML models are in practice often incomplete, and this leads to implementation problems and deviations in the implementation. [Chapter 3]
3. The use of UML models reduces defects in the implementation and reduces the effort for fixing defects. [Chapter 4]
4. The level of detail in UML models, which represents the amount of information that is used to specify models, varies widely across model elements, diagrams, and software projects. Applying higher level of detail in a UML model significantly improves model comprehension. [Chapter 5]
5. Improving the level of detail in UML models (particularly in behavioral diagrams) increases the quality of the implemented system. [Chapter 6]
6. Using UML design metrics for predicting class fault-proneness gives higher accuracy than using code metrics. Applying prediction models that are based on UML design metrics is also potentially more cost-effective. [Chapter 7]
7. To model is to abstract from a reality. With models we capture parts of a reality that are essential to us. Modeling helps us understand real-world problems better.
8. The level of quality needed for a UML model depends on its purpose.
9. A perfect model is not the most cost-effective model.
10. Measurement in software engineering helps to assure repeatable project success.
11. Computer scientists should answer unresolved dogmatic questions more, conduct more experiments and construct more theory.
12. Making good software is like making a good picture. A good picture is about vision, composition, and colors captured early in the camera's viewfinder, not retouched by an editing tool. Likewise, good software is about requirements, architecture, and behaviors captured and designed seamlessly before the implementation, not reworked or patched thereafter.