Summary

Programming is an activity very prone to human error. As more and more features are implemented in a software system by different programmers, progress will often slow to a crawl. It is all too easy for programmers to lose overview of what their code is doing when it is spread across the code base surrounded by the code of others. This can result in bugs and, inevitably, much time will need to be spent on maintenance. This, in turn, results in more expensive software that takes longer to reach the user.

To prevent a large software system from collapsing under its own complexity, its code needs to be well-structured. Manny Lehman (remembered as the Father of Software Evolution) stated the following as his second law of software evolution:

> "As a program is evolved its complexity increases unless work is done to maintain or reduce it."

Ideally we want all code related to a certain *feature* (sometimes called *concern*) to be grouped together in one module —which is called *feature modularization*— and code belonging to different features not be mixed together — which is called *separation of concerns*. But many concerns cannot be easily captured by existing abstractions. They are known as *cross-cutting concerns*. By their very nature their implementation needs to be spread around the code base, so modularization and separation of concerns are still elusive.

This thesis is about *Abstract Delta Modeling (ADM)*, a formal framework developed to achieve modularity and separation of concerns in software.

The software engineering discipline that has the most to gain from those properties is *Software Product Line Engineering (SPLE)*, a relatively new development. To quote van der Linden, Schmid and Rommes:

> "Software product lines represent perhaps the most exciting paradigm shift in software development since the advent of high-level programming languages."

SPLE is concerned with the development and maintenance of *multiple* software systems at the same time, each possessing a different (but often overlapping) set of features — a form of *mass customisation*. This gives rise to an additional need. It is no longer enough that the code for a given feature is separated and modular; it also need to be *composable* and able to deal

gracefully with the presence or absence of other features. We need to be able to make a selection from a set of available features and have the corresponding software mechanically generated for us — a process known as *automated product derivation*. This is another area where ADM can help out.

This thesis is a product of the European HATS project. It presents a formal foundation for the techniques of *delta modeling*, which was the main approach to variability used by HATS. To do this, it employs (among other things) abstract algebra, modal logic, operational semantics and Mealy machines, and lays the bridges between the different disciplines as we go. Its chapters provide a broad overview of the ADM framework and its possibilities, as well as a number of existing practical applications, laying a foundation for further research and development.