

Summary of this Thesis

Today, many consumer products contain ‘small’ computers to control the product internally. Newer products will typically contain more powerful computers than their predecessors, following the same trend as personal and business computers, roughly doubling the capacity every two year according to Moore’s law.

The television was one of the first consumer products with a computer built-in. In 1978, a high-end Philips TV contained a microprocessor with a 1 kilobyte program memory. Ten years later, there were televisions with 64 kilobytes of memory, en another ten years later the size of memory exceeded 1 Megabyte. Other consumer products with built-in computer are video recorders, DVD players and recorders, phones (both fixed and mobile), photo and video cameras, microwave ovens, refrigerators, washing machines, vacuum cleaners, shavers, coffee machines, and the list is growing.

For companies such as Philips this provides the following challenges:

- The complexity of the software in an individual product is growing, and it is becoming increasingly difficult to maintain the desired high quality.
- A company does not sell one product, but rather a family of closely related products. The software should be designed such that as much as possible can be shared between members of the family.
- The market is becoming increasingly dynamic, which makes it important to shorten the time to market, more specifically the development time.
- Companies such as Philips do not produce just one family of products, but rather a number of families (TVs, CD/DVD players, sound amplifiers) which mutually share sets of features. Sharing software between these families reduces development costs even further, but more importantly, it enables the creation of combination products, such as a TV with built-in DVD player.

This thesis builds on three recent trends in science and technology: an increased attention for the *architecture* of software, the invention of techniques to build *software components*, and the systematic creation of *software product lines*. This thesis searches for answers to the following questions:

- Can we make the architecture of software explicit, and at the same time guarantee the consistency between architecture and implementation, so that we can reason about the implementation in terms of the architecture?
- Can we build families and ‘populations’ (read ‘family of families’) of products through the use of software components?
- Can we take advantage of modern techniques to create software components without making our systems more expensive or less optimal in performance?

- What are the consequences of this on the choice of the development process and the required development organization?

This thesis starts by introducing a mathematical technique to model software architectures. Use of this technique enables to check the internal consistency and the consistency with the implementation automatically with the use of a small collection of software tools. This technique, as developed by my colleagues and me has been applied successfully to a large variety of systems. An intrinsic disadvantage of the method is that inconsistencies are usually discovered late in the development process, and solving them is often not cost effective anymore.

Therefore, we introduce an architectural description language from which parts of the implementation can be generated, so that consistency is achieved automatically. At the same time, this language serves as component technology, enabling the creation of different products from the same set of components. The language is designed in such a way that the product's computing resources need not be more expensive, nor that the product is less efficient in performance. Some other contributions of our research are:

- The language supports *independent deployment*: a component need not be developed in a specific context, but can have its own evolution. This requires techniques to implement changes such that newer versions of the component can be used with older versions of other components or vice versa.
- The language emphasizes *composition*, while researchers in software product lines often stress *variation* as the important issue. We see composition as a 'wider' form of variation; the larger the scope of the family (or population) is, the more important composition becomes.
- The classical technique to build more than one products is *configuration management*. We find it important to incorporate management of variation in the architecture, instead of solving it independently.

The thesis describes how the language and method has been applied in Philips to the development of software for mid-range and high-end televisions. This has required an adaptation of the software development process and organization, which used to be optimized for the creation of a single product. We were helped in this by the fact that thinking in families was already starting or common-place in the development of electronics, mechanics and optics.

Finally, we discuss one (be it extended) example of a design pattern that enables composition in a part of the software that was previously considered complex and badly composable. This is by no means the last thing that can be said about composition, and we invite the readers to continue, improve and extend the work as described in this thesis.