The question on the table is if the move towards the reuse paradigm will change the way in which we will generate and use software. This contribution is heavily influenced by the basic assumptions which have been identified at the European Software Institute (ESI), said to be Europe's counterpart to SEI. ESI's mission is to improve the competitiveness of its members.

These assumptions are not guesswork but are results achieved through serious market research. They have been captured in a set of axioms - not in a mathematical sense - describing the future of our discipline.

The contribution presented here begins with a discourse on the history of paradigm changes, which, in a simple picture, can be characterized in a sequence of software ages each dominated by specific approaches in our discipline:

1. The ancient age of mastering the computer: The intuitive use of formalistic languages.
2. The past age of systematic development of today's legacy systems
3. The today's age of architecture based reusability with software components considered to be company assets.
4. The future age of reusability with software components considered to be consumables.

For the sake of this discourse we will concentrate on age 3 by contrasting it against age 4 thus sharpening the understanding of what reusable software means for both software producers and users in economic terms.

Reusable software is assumed to be constituted from reusable objects - of what abstraction level ever. In a simplistic view such objects are conceived to be building blocks for larger software systems. The notion of componentware has become popular for this from of reuse. For European software makers which at 80% are engaged in application construction, reusable components are ideal means for creating applicative solutions fast and at reasonable cost. This, in one sense, is in contrast to the strategy the US follows which today has the lead in libraries of reusable components and mega-size components called standard software products.

The consequence for European software industry is that their focus is more on managing the process of generating solutions of high user-value than on the production of components per se. In our view two classes of application constructions could be,

(a) Architecture driven well engineered constructions of larger and longer-life systems
(b) User driven, ad-hoc configurations of desk-top applications.

The existing expertise in Europe (as far as ESI members are concerned) is on version (a) system. The imagination is, that large scale solutions can be generated by selecting and adapting standard application architectures and then being used as structural skeleton to be filled up with (reusable) components in order to implement the required functionality. This approach presumes the existence both of appropriate architectures and a repository of components, both forming the key assets of the organization producing the software. The models for determining the value of such assets are discussed by John Favaro.

The second major differentiator in following such approach is that such construction by configuration and integration of reusable software needs a minimum level of organizational competence and maturity, in specific w.r.t. process maturity, and again more specific in mastering distributed processes in software engineering. The most demanding scenario actually is
described by the concept of the *Global Software Enterprise* (GSE) a key project at ESI, which as a *scenario project* serve as a catalyst for achieving a new perception of software engineering processes. The assumption is that the software enterprise of the future will be a virtual organization of globally distributed and specialized actors.

The second approach (b) of user defined ad-hoc configurations actually at ESI is discussed under the headlines of the Java Paradigm. Without trying for precision in the definitions of modules, objects or applets, we want to stress that for desk-top applications we can foresee software supply *on demand* by means of user initiated, highly dynamic re-configurations from reusable software constituents fulfilling actual and temporary needs. In such a scenario software will not be considered as a long life asset rather than a *consumable* comparable to money transferred through a current account.

From the point of view of software construction principles such distinction between (a) and (b) may not be significant. From the reuse economics perspective, this will have radical influences on the modeling of the associated processes for defining, creating, maintaining, integrating, configuring and distributing of software components.

What seems to be clear but is not yet achieved: we rather need the process and economical models of the software organization appropriate to the scenario envisaged than we should lose our time in questions on technicalities of reuse. The plan therefore is to discuss the likely scenarios of future application construction.