Diss. ETH No. 14027

The eXtreme Design Approach

A dissertation submitted to the
SWISS FEDERAL INSTITUTE OF TECHNOLOGY
ZÜRICH

for the degree of
Doctor of Technical Sciences

presented by

Adrian Kobler

lic.oec.publ, dipl. Wirtschaftsinformatiker
born February 25, 1964
citizen of Zürich, Switzerland

accepted on the recommendation of

Prof. Dr. M.C. Norrie, examiner
Prof. Dr. S. Spaccapietra, co-examiner

2001

Abstract

Developing software is a complex task involving many technical and non-technical processes. To ensure high quality software systems, adequate software engineering methods must be applied which subdivide the software development process into meaningful phases.

Object-oriented software engineering techniques are suited to managing the complexity of software development and facilitate the decomposition of a software system into extensible and reusable components. This is a very important aspect since it must be taken into account that a software system must be adaptable to new requirements which might not be foreseeable while building it. Thus, it is crucial that the *design* of these components reflect the need for *software evolution.*

Software *prototyping, component-based* software construction and the use of *design patterns* are all means of coping with the complexity and extensibility of software systems. The main problem of designing software systems is to make appropriate decisions about the structure and interoperability of the various components. How should components be linked together? Which relationships between objects of the application domain need to be present? Which properties of an object, such as attributes and methods, must be specified? There are many more questions to be asked during the design process of software systems and components, and many attempts have been made to support this process by guidelines and methods.

In this thesis, we propose the *eXtreme Design* (XD) approach which can be regarded as a supplement to existing methods and models supporting the *design process* of software systems and components with respect to *software evolution* and *persistent data management.* XD is based on the XD meta model together with its algebra combining concepts of conceptual modelling and object-oriented software construction on a meta level.

This makes it possible that classification structures, relationships and object properties can be specified in such a way that they can evolve over time without having to change design and implementation specifications such as class and schema descriptions. Thus, also users can be involved in the evolution process of a software system since they can carry out changes, for example, by introducing new object properties without causing a redesign of the corresponding software components.

XD is especially tailored for designing and implementing those parts of a software system which manage persistent data. We present how XD increases the flexibility and adaptability of these parts of the system by showing how we have applied it in the fields of *prototyping, component modelling* and *rapid information modelling.* We illustrate how we used XD for prototyping a product information system capable of

managing product variants. Further, as an example for component modelling, we describe how XD facilitates the extension of the persistent object management system *OMS Java*. Finally, we outline how XD supports the construction of *information spaces* meeting the demands for various information needs.

Zusammenfassung

Die Entwicklung von Software ist eine komplexe Aufgabe, die sowohl technische als auch nicht-technische Prozesse beinhaltet. Um die Qualität von Softwaresystemen sicherstellen zu können, müssen angemessene Software-Engineering-Methoden angewandt werden, die den Entwicklungsprozess von Software in sinnvolle Phasen unterteilen.

Objekt-orientierte Software-Engineering-Techniken sind sehr geeignet für die Bewältigung der Softwareentwicklungskomplexität und erleichtern die Zerlegung eines Softwaresystems in erweiterbare und wiederverwendbare Komponenten. Dies ist ein sehr wichtiger Aspekt, da in Erwägung gezogen werden muss, dass ein Softwaresystem anpassbar sein sollte an neue Anforderungen, die nicht vorhersehbar waren als das System entwickelt wurde. Daher ist es sehr entscheidend, dass das *Design* dieser Komponenten die Erfordernisse für *Software-Evolution* wiederspiegelt.

Software-*Prototyping*, *komponenten-basierte* Softwareentwicklung und der Gebrauch von *Design-Patterns* sind alles Mittel, mit denen die Komplexität und Erweiterbarkeit von Softwaresystemen gemeistert werden kann. Das Hauptproblem beim Design von Softwaresystemen ist es, eine sachgerechte Entscheidung über die Struktur und das Zusammenwirken der verschiedenen Komponenten zu fällen. Wie sollen die Komponenten miteinander verbunden werden? Welche Beziehungen zwischen Objekten des Anwendungsbereichs müssen vorhanden sein? Welche Objekteigenschaften wie zum Beispiel Attribute und Methoden müssen spezifiziert werden? Es gibt noch viele weiteren Fragen, die während des Design-Prozesses von Softwaresystemen und -Komponenten beantwortet werden müssen, und es wurden schon viele Versuche unternommen, diesen Prozess durch Richtlinien und Methoden zu unterstützen.

In dieser Doktorarbeit schlagen wir den *eXtreme Design* (XD)-Ansatz vor, der als Ergänzung zu existierenden Methoden und Modellen betrachtet werden kann und der den Design-Prozess von Softwaresystemen und -komponenten unterstützt mit Hinsicht auf *Software-Evolution* und Verwaltung von *persistenten* Daten. XD basiert auf dem XD Meta-Modell, das Konzepte des konzeptuellen Modellierens und der objekt-orientierten Softwareentwicklung auf einer Meta-Ebene kombiniert.

Dies macht es möglich, dass Klassifikationsstrukturen, Objektbeziehungen und Objekteigenschaften in einer solchen Art spezifiziert werden können, dass sie sich mit der Zeit weiterentwickeln lassen, ohne dass Design- und Implementationsspezifikationen wie Klassen- und Schemabeschreibungen geändert werden müssen. Daher können auch Benutzer in den Entwicklungsprozess eines Softwaresystems miteinbezogen werden, da sie Änderungen vornehmen können – zum Beispiel um neue

Objekteigenschaften einzuführen – ohne ein Redesign der entsprechenden Softwarekomponenten zu verursachen.

XD ist besonders zugeschnitten für das Designen und Implementieren von denjenigen Teilen eines Softwaresystems, die persistente Daten verwalten. Wir zeigen, wie XD die Flexibilität und Anpassbarkeit dieser Systemteile erhöht und beschreiben, wie wir XD in den Bereichen *Prototyping*, *Komponentenmodellierung* und *Informationsmodellierung* angewandt haben. Wir präsentieren, wie wir XD für das Prototyping eines Produktinformationssystems, das fähig ist Produktvarianten zu verwalten, gebraucht haben. Zudem beschreiben wir, als ein Beispiel für Komponentenmodellierung, wie XD die Erweiterung des persistenten Objektverwaltungssystems *OMS Java* erleichtert. Schliesslich stellen wir kurz dar, wie XD die Konstruktion von *Informationsräumen*, die den Anforderungen nach verschiedenartigen Informationsbedürfnissen nachkommen, unterstützt.