PDSC: eine Methode zur Software-Entwicklung

Horst Lichter, Institut für Informatik
ETH-Zentrum, CH 8092 Zürich[1]

Zusammenfassung

Die Arbeit beschreibt die Software-Entwicklung nach dem PDSC-Ansatz (**P**rogram **D**evelopment by **S**tepwise **C**ompletion). Dieser Ansatz ist dadurch gekennzeichnet, daß man bei der Software-Entwicklung, ausgehend von einer Architekturbeschreibung, über einen System-Prototyp zum Zielsystem gelangt. Die Methode PDSC wird ausführlich in [LUD 87] beschrieben. Diese Arbeit geht insbesondere auf die Aspekte *Behandlung von Unvollständigkeiten* und *Software-Entwicklung mit Prototypen* ein. Weiterhin wird die Vorgehensweise bei der Software-Entwicklung mit PDSC beschrieben und die Programm-Entwicklungsumgebung skizziert, die für diese Methode notwendig ist.

Einleitung

Software wird heute meistens nach einem *Phasenplan* entwickelt. Charakteristisch für diesen Entwicklungsprozeß ist die Tatsache, daß das Produkt als Ergebnis des sequentiellen Durchlaufs durch die Entwicklungsphasen (Anforderungen finden, spezifizieren, entwerfen,...) entsteht. Ein großer Nachteil dieser Vorgehensweise besteht darin, daß Änderungen der Anforderungen oder Fehler im Entwurf erst beim fertiggestellten Produkt sichtbar werden. Korrekturen sind dann, wenn überhaupt, nur schwer möglich und sehr aufwendig.

Eine andere Vorgehensweise bei der Software-Entwicklung wird als *Prototyping* bezeichnet. Der Prototyping-Ansatz wird u.a. in [BUD 84] und [BUD 87] näher diskutiert. Dieser Ansatz ist dadurch gekennzeichnet, daß bereits sehr früh in der Entwicklung ein lauffähiger Prototyp des geplanten Systems erstellt wird (rapid prototyping). An diesem können nun die Systemanforderungen überprüft werden. Notwendige Änderungen sind am Prototyp leicht möglich und können vergleichsweise schnell durchgeführt werden. Erst, wenn das Verhalten des Prototyps den gestellten Erwartungen entspricht, wird das Zielsystem realisiert. Der Prototyp wird dazu nicht mehr gebraucht und weggeworfen. Ein Manko dieses Ansatzes ist die Tatsache, daß die bei der Entwicklung des Prototyps gewonnene Information nicht explizit für die Konstruktion des Zielsystems verwendet wird.

Mit der Definition der Entwicklungsmethode PDSC versuchen wir u.a. die Vorteile der beiden geschilderten Vorgehensweisen zu vereinen.

Ziele der Methode PDSC

Mit der Definition unserer Methode zur Software-Entwicklung sollen hauptsächlich die folgenden Ziele erreicht werden:

1. Behandlung von Unvollständigkeiten während der Software-Entwicklung.
2. Software-Entwicklung mit Prototypen und Verwendung der beim Prototyping gewonnenen Information zum Bau des Zielsystems.

---

[1]Die in diesem Bericht präsentierten Ergebnisse und Ideen sind in der Gruppe Software Engineering der ETH Zürich enstanden. Maßgeblich daran beteiligt sind J. Färberböck (jetzt SBG Zürich), J. Ludewig und H. Matheis.

3. Unterstützung beim "Programming in the Large".
4. Konstruktion von wartungsfreundlicher, effizienter Software.

Die beiden ersten Zielsetzungen werden nachfolgend ausführlich erläutert.

1. Behandlung von Unvollständigkeiten

Bei der Konstruktion von Software wird der Entwickler immer Situationen vorfinden, in denen das System nur unvollständig beschrieben ist. Diese Unvollständigkeiten sind zwangsläufig dadurch gegeben, daß zu Beginn der Entwicklung, aber auch während der Entwicklung selbst, nicht alle notwendigen Komponenten des Systems identifiziert oder sogar hinreichend detailliert beschrieben werden können. Der Extremfall der Unvollständigkeit des Systems liegt beim Projektstart vor. Zu diesem Zeitpunkt ist außer vagen Ideen und Wünschen, die durch ein Software-System verwirklicht werden sollen, nichts bekannt.

Die Aufgabe der Software-Entwickler besteht nun darin, diese Unvollständigkeiten zu beseitigen und zum gewünschten System zu gelangen. Die Unvollständigkeiten im System sind also erst dann beseitigt, wenn es ganz implementiert ist.

Es werden prinzipiell drei Arten von Unvollständigkeiten unterschieden [LUD 85]:

1. *fehlende Teile*
   Beispiel: es wurde noch nicht festgestellt, daß zum Funktionieren des Systems gewisse Komponenten notwendig sind.
2. *Unbestimmtheit von Teilen*
   Beispiel: man weiß zwar, daß man eine Komponente braucht, kann aber deren Leistungsumfang noch nicht beschreiben.
3. *fehlende Detaillierung*
   Beispiel: es ist noch nicht festgelegt, wie eine Komponente aufgebaut ist.

Weiterhin können Unvollständigkeiten vorkommen, die nicht eindeutig einer der aufgezählten Arten zugeordnet werden können, sondern als eine Mischung dieser Arten anzusehen sind.
Der Entwickler benötigt folglich eine Möglichkeit Unvollständigkeiten, die während der Software-Erstellung auftreten, behandeln zu können. Dazu muß er in der Lage sein, das geplante System initial als *Software-Skizze* zu beschreiben, in der im allgemeinen alle Arten von Unvollständigkeiten auftreten. Er muß also Möglichkeiten besitzen, um Unvollständigkeiten *ausdrücken* und während der Entwicklung *verwalten* und *auflösen* zu können. Diese Möglichkeiten sollen ihm mit der Methode PDSC angeboten werden.

2. Software-Entwicklung mit Prototypen

Wie bereits in der Einleitung erwähnt, wird bei der Software-Entwicklung mit Prototypen bereits frühzeitig ein Prototyp des zu entwickelnden Systems erstellt. Dieser Prototyp braucht weder effizient noch leicht wartbar zu sein (quick and dirty). Seine Aufgabe besteht darin, das Verhalten des Systems zu demonstrieren, damit die Anforderungen an das System überprüft werden können. Notwendige Modifikationen werden schnell am Prototyp durchgeführt und der Prototyp wird anschließend einer erneuten Bewertung unterzogen. Zeigt der Prototyp das gewünschte Verhalten, so dient er nur noch als Teil der Spezifikation des Zielsystems, wird selber aber nicht weiterverwendet (throw-it-away-Prototyping).
Diese Vorgehensweise besitzt gegenüber der traditionellen Software-Entwicklung nach

einem Phasenplan wesentliche Vorteile. Die folgenden Vorteile sind unserer Meinung nach besonders wichtig:

- frühe Realisierung eines lauffähigen Modells des Zielsystems
- frühzeitige Überprüfung der Systemanforderungen
- Erleichterung der Integration des Auftraggebers/Benutzers in den Entwicklungsprozeß
- Verkürzung der "Durststrecke" zwischen Projektstart und erstem sichtbaren und lauffähigen Ergebnis

Diesen Vorteilen stehen Probleme gegenüber, die sich bei der Software-Entwicklung mit Prototypen ergeben können. Zwei dieser Probleme sind:

- Wann soll die Arbeit am Prototyp beendet werden und die Konstruktion des eigentlichen Systems begonnen werden?
- Wie kann die im Prototyp enthaltene Information explizit beim Bau des Zielsystems weiterverwendet werden?

Aus den Vorteilen und den geschilderten Problemen des Prototyping-Ansatzes folgt für uns, daß dem Entwickler mit der Methode PDSC die Möglichkeit angeboten werden muß, das System sowie Systemteile als Prototyp zu realisieren, um alle Vorteile dieses Ansatzes ausnutzen zu können. Weiterhin soll die Methode aber auch gewährleisten, daß ein Großteil der Information, die bei der Entwicklung des Prototyps gewonnen wurde, in die Konstruktion des Zielsystems einfließt.

Vorgehensweise bei der Software-Entwicklung mit PDSC

Eine wichtige Rolle bei der Software-Entwicklung spielen die für die unterschiedlichen Aufgaben verwendeten Sprachen. Dies gilt ebenfalls bei der Anwendung von PDSC. *Wir* verwenden vor allem drei Sprachen

- *ADL* zur Beschreibung der Software-Architektur
- *Smalltalk-80* zum Prototyping
- *Modula-2* oder *Ada* zur Codierung des Zielsystems

Welche Anforderungen PDSC-Sprachen erfüllen müssen, und warum wir die oben genannten Sprachen einsetzen, ist in [LUD 87] beschrieben. Die Anwendung von PDSC ist aber nicht abhängig von diesen Sprachen. Sie stehen als Repräsentanten von Sprachklassen. Es können ebensogut andere Sprachen verwendet werden, soweit diese für die speziellen Aufgaben bei PDSC geeignet sind.

Ausgangspunkt für den Einsatz von PDSC ist eine informale Spezifikation. Bei der Aufnahme dieser Spezifikation und deren Strukturierung bietet PDSC keine Unterstützung. Auf der Basis der strukturierten Spezifikation wird die Software-Architektur entworfen. Dieses geschieht mit der Architektur-Sprache ADL und den dazu gehörenden Werkzeugen. Die Software-Architektur legt die Bestandteile der Software, deren Aufgaben und Beziehungen fest. Diese Teile sind sowohl Programm-Module als auch andere Dokumente, die für das System relevant sind. Abbildung 1 zeigt ein Beispiel einer Software-Architektur.

Aufgrund der definierten Software-Architektur wird ein System-Prototyp erstellt. Dies geschieht zweistufig. Im ersten Schritt werden die beschriebenen Programmkomponenten mit einem Transformator als Klassenrahmen im Smalltalk-System abgelegt. Diese können anschließend in beliebiger Reihenfolge verfeinert und prototypisch implementiert werden. Somit entsteht ein erstes lauffähiges System, das einer Erprobung und Analyse unterzogen werden kann. Komponenten, die bereits in der Zielsprache vorliegen, brauchen natürlich nicht mehr als Prototyp implementiert zu werden, sondern können unverändert oder auf die speziellen Zwecke angepaßt im System-Prototyp verwendet werden.

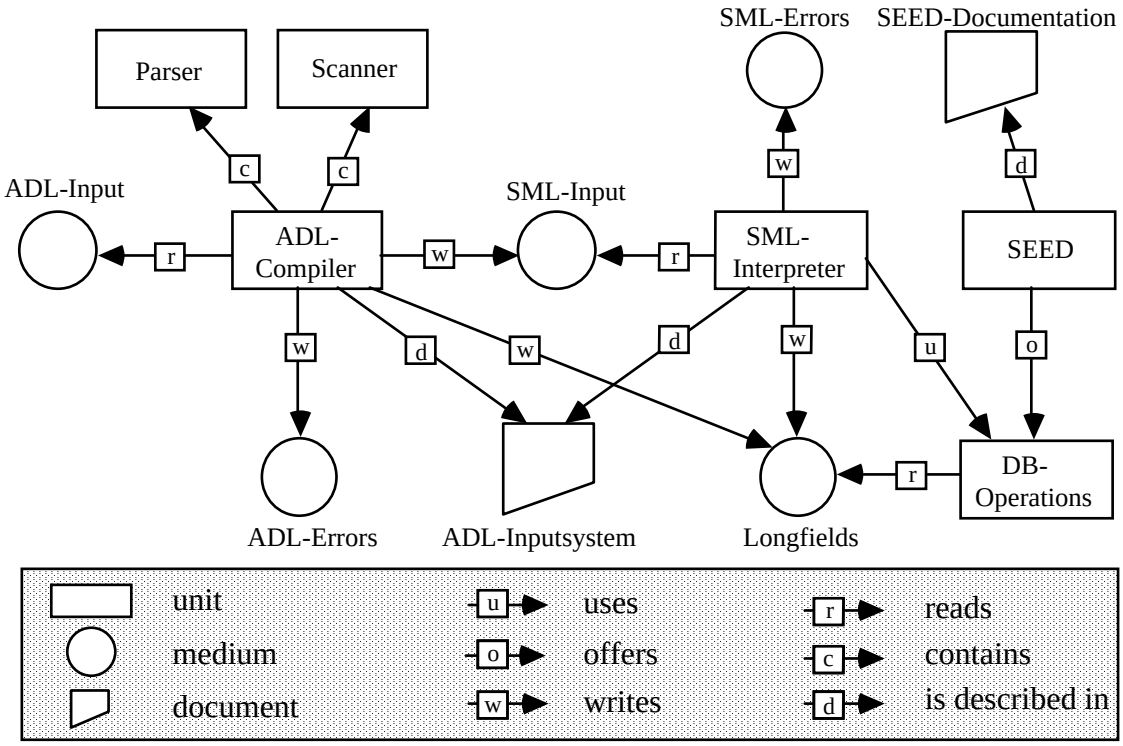

Abbildung 1: graphische Darstellung einer noch unvollständig beschriebenen Software-Architektur.

In dem nun beginnenden Entwicklungsprozeß vom Prototypen zum Zielsystem hin, den wir als "development by stepwise completion" bezeichnen, ist es möglich,

- kritische Teile in der Zielsprache zu implementieren, um deren Realisierbarkeit zu zeigen,
- Komponenten, deren Leistung oder Verhalten im Prototyp als befriedigend bewertet wurde, in der Zielsprache zu implementieren,
- Unvollständigkeiten in der Architekturbeschreibung zu beseitigen, indem vage Bestandteile konkretisiert oder fehlende Teile ergänzt werden, und den Prototyp neu zu konfigurieren und zu analysieren.

Während dieses gesamten Prozesses bleibt der Prototyp lauffähig. Der Prototyp besteht also während dieser Zeit aus Komponenten, die in den unterschiedlichen Entwicklungsstufen (Beschreibungszustand, Prototyp-Implementierung, Zielimplementierung) vorliegen. Die Entwicklung ist dann beendet, wenn alle Teile des Systems in der Zielsprache codiert sind.

Durch diese Vorgehensweise ist gewährleistet, daß die im Prototyp enthaltene Information beim Übergang zum Zielsystem nicht verloren geht. Es handelt sich hier vor allem um die Strukturinformation, die in der Architekturbeschreibung gesammelt wird.

Die PDSC-Entwicklungsumgebung

Die beschriebene Vorgehensweise ist nur mit der Unterstützung einer Programmentwicklungsumgebung durchführbar.

Zentraler Bestandteil der gesamten Entwicklungsumgebung ist eine Progammentwicklungs-Datenbank. Sie nimmt alle bei der Programmentwicklung anfallenden Dokumente auf und verwaltet sie. Wir glauben, daß eine Entity-Relationship-Datenbank besonders zu diesen Zwecken geeignet ist, und verwenden eine speziell dafür entwickelte Datenbank [GLI 85], [GLI 86].

Die PDSC-Entwicklungsumgebung enthält neben dieser Datenbank hauptsächlich die folgenden drei Werkzeuggruppen.

1. **sprachspezifische Werkzeuge**
   Da wir für die drei Haupttätigkeiten bei PDSC (Architektur beschreiben, Protoyp implementieren, Zielsystem codieren) drei Sprachen verwenden, sind Werkzeuge zum Arbeiten mit diesen Sprachen notwendig.

   - Eingabewerkzeuge und Editoren zur Beschreibung der Software-Architektur in der Sprache ADL (*ADL-Editor* und *ADL-Eingabesystem*). Mit diesen können Architekturen sowohl graphisch als auch textuell bearbeitet werden. Die Werkzeuge arbeiten direkt auf der Datenbank, so daß eine konsistente Beschreibung der Architektur gewährleistet werden kann.

   - Eine *Prototyp-Entwicklungsumgebung* im Smalltalk-System. Sie erleichtert eine schnelle Implementierung der Prototypen. Den Kern dieses Werkzeugs bildet ein modifizierter System-Browser. Er enthält die in ADL beschriebenen Systemkomponenten, bietet die Möglichkeit, Unstimmigkeiten zwischen der ADL-Beschreibung und der Prototyp-Implementierung festzustellen und erleichtert die nach einer Veränderung der ADL-Beschreibung notwendig gewordenen Änderungen am Prototyp.

   - Eine Programmentwicklungsumgebung für die Zielsprache *Ada* oder *Modula-2*.

2. **methodenunterstützende Werkzeuge**
   Sie dienen vor allem dazu, den Übergang von der Struktur-beschreibung der Software zum Prototyp und vom Prototyp zur Zielimplementierung zu unterstützen. Hierzu gehören:

   - das Werkzeug *ADL-ST80*; es generiert aus der in der Datenbank enthaltenen Architekturbeschreibung Klassenrahmen und stellt sie dem Smalltalk-System zur Verfügung.

   - die Werkzeuge *ST80-M2* und *ST80-ADA*; sie generieren aus den Prototypkomponenten Modul- bzw. Paketrahmen für die jeweilige Zielsprache.

3. **prototypspezifische Werkzeuge**
   Hierunter fallen hauptsächlich die folgenden Komponenten:

   - ein *Prototyp-Konfigurator*. Er baut aus den Software-Komponenten, die in den unterschiedlichen Entwicklungs-zuständen sind, einen ablauffähigen System-Prototyp. Dazu wertet er die Architekturbeschreibung aus und wählt von jeder Komponente die präziseste verfügbare Ausprägung.

- ein *Prototyp-Simulator*. Mit ihm kann der vom Konfigurator erstellte System-Prototyp erprobt und analysiert werden. Dabei muß die Möglichkeit bestehen, daß Unvollständigkeiten (z.B. ein noch nicht implementiertes Modul) zur Laufzeit ausgeglichen werden können. Dies kann mittels Standardaktionen oder durch interaktives Eingreifen geschehen.

Die nachfolgende Abbildung zeigt eine schematische Darstellung der Komponenten der PDSC-Umgebung.

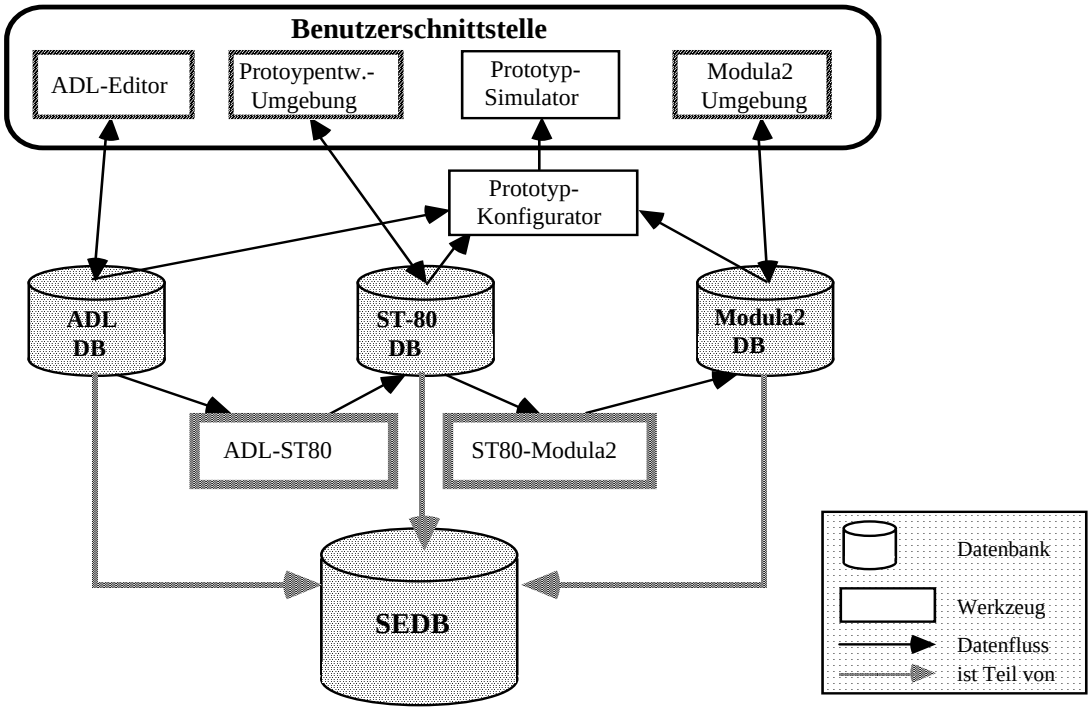

Abbildung 2: Die PDSC-Entwicklungsumgebung. Die Werkzeuge sind speziell auf die von uns verwendeten Sprachen abgestimmt und sind nach der vorher beschriebenen Klassifizierung graphisch unterschiedlich dargestellt. Die drei abgebildeten Teil-Datenbanken sind nur aus optischen Gründen separat aufgeführt. Sie sind logisch und physisch in der Software-Entwicklungsdatenbank (SEDB) enthalten.

Stand der Arbeit, Pläne

Mit der Definition der Methode PDSC haben wir versucht, insbesondere die Aspekte
- Behandlung von Unvollständigkeiten und
- Software-Entwicklung mit Prototypen

in den Software-Entwicklungsprozess zu integrieren. Das bedeutet konkret:

- sukzessives Vervollständigen
  - vom Vagen, Unvollständigen zum Konkreten, Vollständigen,
  - vom Ineffizienten zum Effizienten,
  - vom Ungeprüften zum Geprüften
- Beibehalten aller Vorteile des Prototyp-Ansatzes
- Verwenden der im Prototyp enthaltenen Information bei der Realisierung des Zielsystems.

Die ersten Ansätze und Ideen zu PDSC begannen im Herbst 1986. Wir haben bis jetzt folgende Bereiche bei PDSC bearbeitet:

- Realisierung der Software-Entwicklungsdatenbank auf der Grundlage eines geeigneten Basissystems.
- Definition der ersten Version der Archtitektursprache ADL.
- Konstruktion eines flexiblen ADL-Eingabesystems und Editors.
- Konzeption des Übergangs von der Architekturbeschreibung zur Prototyp-Implementierung und Realisierung des dazu notwendigen Werkzeugs ADL-ST80.

Bis Ende dieses Jahres sind folgende Arbeiten geplant:

- Realisierung einer Schnittstelle zwischen der Prototyp-Sprache Smalltalk-80 und der Zielsprache Modula-2. Diese ist eine Voraussetzung für den Prototyp-Simulator.
- Erweiterung der Datenbank um eine Versionen-, Varianten- und Konfigurationenverwaltung.
- Überarbeitung unserer Architektursprache ADL
- Konzipierung und Implementierung einer Prototyp-Entwicklungsumgebung für Smalltalk-80.

Literatur

[BUD 84] Budde R., Kuhlenkamp K., Mathiassen L., Züllighoven H. : **Approaches to Prototyping**. Springer Verlag, Berlin, 1984.

[BUD 87] Budde R., Kuhlenkamp K., Sylla K.H., Züllighoven H.: **Konzepte des Prototyping**. in Requirements Engineering´87, GMD-Studien Nr 121, Gesellschaft für Mathematik und Datenverarbeitung mbH, Sankt Augustin, 1987, pp 77-93.

[DIE 87] Diederich L.G., Milton J.: **Experimental Prototyping in Smalltalk-80**. IEEE Software, May 1987, pp 50-64.

[GLI 85] Glinz M., Huser H.J., Ludewig J. : **SEED - A database system for software engineering environments.** in Blaser, Pistor (Hrsg.): Datenbanksysteme für Büro, Technik und Wissenschaft, Informatik-FB 94, Springer, S.121-126.

[GLI 86] Glinz M., Ludewig J.: **SEED - a DBMS for Software Engineering applications based on the Entity Relationship approach**. to appear in G. Wiederhold: The Second Intern. Conf. on Data Engineering. Los Angeles, CA, February 5-7, 1986.

[LUD 87] Ludewig J., Färberböck H., Lichter H., Matheis H., Wallmüller E.: **Software-Entwicklung durch schrittweise Komplettierung.** in Requirements Engineering´87, GMD-Studien Nr 121, Gesellschaft für Mathematik und Datenverarbeitung mbH, Sankt Augustin, 1987, pp 113-124.

[LUD 85] Ludewig J., Glinz M., Matheis H.: **Software-Spezifikation durch halbformale, anschauliche Modelle**. in H.R. Hansen: GI/OCG/ÖGI- Jahrestagung 1985, Informatik-FB 108, Springer-Verlag, Berlin usw., pp193-204.