�☞DFD (Data Flow Diagram)

쉽게 말해 자료의 출발지와 목적지를 그림으로 표시한 것이다.

데이터가 소프트웨어 내의 각 프로세스를 따라 흐르면서 변환되는 모습을 나타낸 그림으로 소프트웨어 및 정보시스템의 분석과 설계에서 매우 유용하게 사용되는 다이아그램이다.

'데이터 흐름도' 또는 '자료 흐름도' 라고도 한다.

☞DFD의 구성요소

DFD의 구성요소 4가지는 프로세스, 데이터흐름, 데이터저장소, 외부엔티티이며, 표기법에 따라 표현하는 그림의 모양이 달라진다.Yourdon과 DeMarco에 의해 주장된 표기법을 설명한다.

1. 프로세스(Process) - 원(Bubble)

입력되는 데이터를원하는 데이터로 변환하여 출력시키기 위한 과정이다.

도형적 표기형태는 원(Bubble)과 원안의 이름으로 표현한다.

원안에 기록하는 이름은프로세스가 수행하는 일또는프로세스를 수행하는 행위자를 기술한다.

자체적으로는 데이터를 생성할 수 없고 항상입력되는 데이터가 있어야 한다.

항상 새로운 가치를 부가해야 한다.



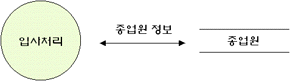
2. 데이터흐름(Data Flow) - 화살표

DFD의 구성요소간 인터페이스를 나타낸다.

대부분의데이터흐름은 프로세스 사이를 연결하지만, 데이터 저장소(Data Store)로부터의 데이터흐름을 나타내기도 한다.

명칭이 부여거나 부여되지 않은 화살표로 표시한다. 단, 후속작업의 참조를 위해 되도록 명칭이 부여되는 것이 바람직하다.

서로 다른 데이터 흐름에는동일한 이름을 부여하지 않는다.



3. 데이터저장소(Data Store) - 두 평행선

추후 엑세스를 위해 데이터를 저장하는수동적 객체를 말한다.

액터와는 달리 자신의 액션을 취하지 않으며 데이터 저장이나 엑세스 요구에 반응할 뿐이다.

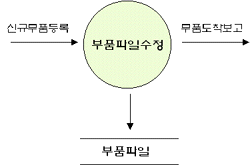
데이터저장소(Data Store)는저장되어 있는 정보 집합이다.

데이터저장소는 테이프, 디스크, 카드 데이타, 캐비넷의 인덱스화일 등일 수도 있으며, 때로는 휴지통일 수도 있다.

데이터저장소는단순한 데이터의 저장을 나타내는 것이지 데이터의 변동을 표시하는 것은 아니다.

데이터흐름을 표시함으로서 데이터의 입출력을 나타낸다.

표기법은 단순하게 두 개의 직선 즉, 평행선으로 나타내고, 평행선 안에 데이터저장소의 명칭을 부여한다.



4. 외부엔티티(External Entity) - 사각형

프로세스 처리과정의 데이터발생의 시작 및 종료를 나타낸다.

어떤 기업의내적인(Inside) 엔티티는 관리, 부서, 기능, 시스템등을 포함하며, 기업외적인(Outside) 엔티티는 고객, 거래처, 공공기관, 외부시스템등을 포함한다.

데이터 흐름도상에서 프로세스(Process)와의 상호관련성을 표시하며, 일반적으로DFD 범위 밖에사각형 형태로 표시한다.

액터는 데이터를 생성, 소비 함으로서 데이터 흐름도를 주도하는 활성 객체이다.

액터는 데이터 흐름도의 입력과 출력에 붙는다. 즉, 흐름도의 경계에 놓이게 되며, 소스(source)나 싱크(sink)로서 데이터의 흐름을 중단시킨다.

예를 들어, 프로그램의 사용자, 써모셋(thermostat), 서보 모터 등이다.



☞DFD의 작성방법

업무를 분석하여 프로세스에 대한모든 입출력 데이터흐름을 식별한다.

흐름상 필요하거나 제공되어야 할외부엔티티를 정의한다.

입력으로부터 출력으로, 출력으로부터 입력으로, 또는 중간 지점부터의데이터흐름을 식별한다.

모든 접속관계 데이터흐름에 주의깊게명칭(혹은 자료 내역)을 부여한다.

프로세스에 대해 입출력 데이터흐름의 명칭에 따라이름을 부여한다.

프로세스에 관련된데이터저장소를 정의한다.

검토하고 보완한다.

상위레벨 DFD완성 후, 하위 레벨의 DFD로 분할하여 최하위 레벨까지 그린다.(일반적��로 3번 반복)

|  |
| --- |
| DFD(Data Flow Diagram)는 데이터가 소프트웨어 내의 각 프로세스를 따라 흐르면서 변환되는 모습을 나타낸 그림으로 소프트웨어 및 정보시스템의 분석과 설계에서 매우 유용하게 사용되는 다이아그램이다. 데이터 흐름도 또는 자료 흐름도라고 칭하기도 한다. DFD는 시스템의 모형화 도구로서 가장 보편적으로 사용되는 것 중의 하나이며 데이타에 비해 기능이 매우 복잡하고 중요할 경우에 매우 유용하게 사용될 수 있다.    [그림 1] DFD의 예 |

|  |
| --- |
| **1. 구조적 방법론과 DFD** |
| DFD는 이른바 구조적 방법론을 대표하는 다이아그램이다. 구조적 분석/설계 혹은 구조적 방법론은(DFD의 모습처럼 "자료흐름지향 방법론"이라고도 한다) 1968년 Dijskstra가 GOTO文의 폐해를 들어 구조적 프로그래밍(Structured Programming)의 개념을 소개하면서 시작되었다. 모든 방법론은 항상 프로그래밍으로부터 문제가 제기되고 이를 분석/설계에도 적용시켜보자는식으로 발전한다. 구조적 분석/설계에서의 데이터는 기능(또는 프로세스) 사이에서 주고 받는 형태로만 나타나며 기능의 계속된 분해와 이의 분석을 통해서만 데이터가 나타나게 된다. 이 후 70년대를 거치면서 거의 대부분의 소프트웨어 개발은 '구조적'이라는 용어에 둘러싸이다시피 했으며 소프트웨어 공학에서도 가장 빈번한 이슈였다. 따라서 소프트웨어 개발과정의 중간 산출물로 DFD를 비롯한 DD(Data Dictionary), 구조도(Structured Diagram) 등이 많이 사용되었다. 그러나 소프트웨어의 규모와 복잡도가 커지고 기업에서 보다 많은 소프트웨어와 컴퓨터 시스템을 사용하게 됨에 따라 자료흐름 위주의 분석과 설계는 한계에 부닥치게 된다. 정보시스템을 비롯한 MIS관련 소프트웨어는 많은 량의 데이터를 관리할 필요가 생기고 그 구조의 변경이 프로그램의 유지보수에 큰 영향을 미치기 때문이다. 따라서 프로세스 위주의 DFD보다는 데이터에 대한 분석/설계가 더 안정적이다. 최근의 정보시스템을 포함한 소프트웨어의 설계에서는 DFD의 중요성은 그만큼 덜 해지고 ERD(Entity Relationship Diagram) 같은 데이터 분석 도구가 더 유용하게 사용되고 있다. 정보공학(Information Engineering)에서는 DFD를 거의 강조하지 않으며 데이터와 프로세스가 동시에 나타난다는 이유로 검증용 도구로 사용되는 정도에 그치고 있다. 그러나 실제 소프트웨어 개발 프로젝트에서는 아직도 DFD와 그에 따른 산출물들이 절찬리에 사용되고 있는데, 그 이유는 아마도 많은 개발자들이 이에 익숙해져 있고, 실제로 다른 다이아그램에 비해 DFD가 표현하고 이해하기가 매우 쉬운편이기 때문이 아닐까 생각된다. |

|  |
| --- |
| **2. DFD 작성의 이익** |
| 다른 분석/설계용 문서 산출물들과 마찬가지로 다음과 같은 잇점을 가져다 줄 수 있다.   * 현업사용자의 업무 및 요구사항을 쉽게 문서화 할 수 있다. * 현업사용자와 분석가(또는 개발자) 사이의 의사소통을 위한 공용어의 역할을 한다. * 일관성 있고 정확한 사용자의 요구사항을 파악할 수 있는 요구분석용 도구의 역할을 한다. |

|  |
| --- |
| **3. DFD의 특성** |
| DFD는 다른 다이아그램과 구별되는 다음과 같은 특징들을 갖는다.   * 도형으로 그려지는 그림 중심의 표현이다. * 다차원적(Multidimensional)이다. * 데이터(자료)의 흐름에 중심을 두는 분석용 도구이다. * 제어(Control)의 흐름은 중요시 하지 않는다. |

|  |
| --- |
| **4. DFD의 구성요소** |
| DFD의 구성요소로는 프로세스, 데이터흐름, 데이터저장소, 외부엔티티 등이 있으며 표기법에 따라 표현하는 그림의 모양이 달라진다. 여기서는 Yourdon과 DeMarco에 의해 주장된 표기법을 기준으로 설명한다.  **4.1 프로세스(Process)**   * 프로세스는 입력되는 데이터를 원하는 데이터로 변환하여 출력시키기 위한 과정으로 도형적 표기형태로는 원(Bubble)과 원안의 이름으로 표현한다. * 원안에 기록하는 이름은 아래에 그림과 같이 프로세스가 수행하는 일 또는 프로세스를 수행하는 행위자를 기술한다. * 프로세스는 자체적으로는 데이터를 생성할 수 없고 항상 입력되는 데이터가 있어야 한다. * 프로세스는 항상 새로운 가치를 부가해야 한다.     [그림 2] Process  **4.2 데이터흐름(Data Flow)**   * 데이터흐름(Data Flow)은 DFD의 구성요소들간의 인터페이스를 나타낸다. * 대부분의 데이터흐름은 프로세스들 사이를 연결하지만, 데이터 저장소(Data Store)로부터의 데이터흐름을 나타내기도 한다. * 데이터흐름은 명칭이 부여거나 부여되지 않은 화살표로 표시한다. 단, 후속작업들의 참조를 위해 되도록 명칭이 부여되는 것이 바람직하다. * 서로 다른 데이터 흐름에는 동일한 이름을 부여하지 않는다.     [그림 3] Data Flow  **4.3 데이터저장소(Data Store)**  데이터 스토어는 추후의 억세스를 위해 데이터를 저장하는 수동적 객체를 말한다. 데이터 스토어는 액터와는 달리 자신의 액션을 취하지 않는다. 다만 데이터 저장이나 억세스 요구에 반응할 뿐이다.  비록 많은 경우 데이터 흐름은 특정 값을 나타내지만 어떤 경우 데이터 흐름이 객체를 나타낼 때도 있다. 또한 한 객체를 하나의 값(single value)로 볼 수도 있고 많은 값을 포함하는 데이터 스토어로 볼 수도 있다.   * 데이터저장소(Data Store)는 저장되어 있는 정보 집합이다. * 데이터저장소는 테이프, 디스크, 카드 데이타, 캐비넷의 인덱스화일 등일 수도 있으며, 때로는 휴지통일 수도 있다. * 데이터저장소는 단순한 데이터의 저장을 나타내는 것이지 데이터의 변동을 표시하는 것은 아니다. * 데이터흐름을 표시함으로서 데이터의 입출력을 나타낸다. * 데이터 흐름도에서 데이터저장소를 나타내는 표기법은 단순하게 두개의 직선 즉, 평행선으로 나타내고, 평행선 안에 데이터저장소의 명칭을 부여한다.     [그림 4] Data Store  **4.4 외부엔티티(External Entity)**  액터는 데이터를 생성, 소비 함으로서 데이터 흐름도(data flow graph)를 주도하는 활성 객체이다. 액터는 데이터 흐름도의 입력과 출력에 붙는다. 즉 보통 데이터 흐름도의 경계에 놓이게 되며, 소스(source)나 싱크(sink)로서 데이터의 흐름을 중단시킨다. 예를 들어, 프로그램의 사용자, 써모셋(thermostat), 서보 모터 등이다. 액터가 발생시키는 액션은 동적모델에 표현된다.   * 외부엔티티는 프로세스 처리과정의 데이터발생의 시작 및 종료를 나타낸다. * 어떤 기업의 내적인(Inside) 외부엔티티는 관리, 부서, 기능, 시스템등을 포함하며, 기업 외적인(Outside) 외부엔티티는 고객, 거래처, 공공기관, 외부시스템등을 포함한다. * 외부엔티티는 데이터 흐름도상에서 프로세스(Process)와의 상호관련성을 표시하며, 일반적으로 DFD 범위 밖에 사각형 형태로 표시한다.     [그림 5] External Entity |

|  |
| --- |
| **5. 작성방법** |
| DFD의 작성방법은 다음과 같다.   * 업무를 분석하여 프로세스에 대한 모든 입출력 데이터흐름을 식별한다. 그리고 업무의 주변 경계에 그들을 표시한다. * 데이터흐름상 필요하거나 제공되어야 할 외부엔티티를 정의한다. * 입력으로부터 출력으로, 출력으로부터 입력으로, 또는 중간 지점부터의 데이터흐름을 식별한다. * 모든 접속관계 데이터흐름에 주의깊게 명칭(혹은 자료 내역)을 부여한다. * 프로세스에 대해 입력 데이터흐름과 출력 데이터흐름의 명칭에 따라 이름을 부여한다. * 프로세스에 관련된 데이터저장소를 정의한다. * 검토하고 보완한다. * 상위레벨 DFD완성후 다음 하위 레벨의 DFD로 분할하여 최하위 레벨까지 그린다. * 데이터 흐름도의 규모가 너무 커서 한 장의 종이에 그릴 수 없을 때는 시스템을 서브시스템(Subsystems)들로 분할한다. 분할된 서브시스템들의 규모가 클때는 다시 분할을 계속한다. 이렇게 세분화를 계속하여 마지막에는 데이타 흐름도를 단순한 기능들만으로 그릴 수 있는 단계까지 분할한다. (일반적으로 레벨3까지면 적당하다)   **5.1 배경도(Context Diagram)**  배경도는 분석하고자 하는 시스템과 외부 세계와의 접속관계를 식별하기 위한 것으로 시스템 분석의 범위를 결정하게 된다.  **5.2 최하위단계 데이터 흐름도**  최하위 단계의 DFD는 DFD 내의 모든 프로세스가 더 이상 하위 단계의 DFD로 분할되지 않는 데이터 흐름도를 말하며, 모든 프로세스들이 명세서를 갖는 기본단위인 것을 말한다.  **5.3 상하관계**  다음 그림은 분할된 데이터 흐름도의 상하관계를 보여주는데 상위 단계의 번호1로 표시된 프로세스가 다시3개의 하위 단계의 프로세스들로 분할되는 것을 보여주고 있다. 하위 단계에서는 분할된 프로세스들과 그들 간의 접속관계를 보여주며 상위 단계의 프로세스와 데이터흐름을 더욱 상세히 설명한다. 하위단계의 데이터 흐름도에서는 프로세스 분할뿐만 아니라 데이터흐름의 변환과정도 자세히 나타나는데, 그림에서는 상위 단계의 데이터흐름 A가 B와 C로 변환되는 과정을 상세히 설명하고 있다. 그러나 하위 단계는 상위 단계와 동일한 데이터변환을 하는 것이므로 순수입력과 순수출력은 상위와 하위 단계 모두 일치한다. 그림에서 또 유의해야 할 것은 상위 단계의 프로세스1이 분할된 하위 단계의 프로세스들은 1.1, 1.2, 1.3 등과 같은 번호를 부여받는다. 마찬가지로 하위 단계의 프로세스 1.2가 다시 분할된다면 그것의 하위 단계 데이타 흐름도는 다시 1.2.1, 1.2.2, 1.2.3 등과 같이 된다.    [그림 6] DFD의 분할 |

|  |
| --- |
| **6. 작성규칙** |
| DFD 만큼 애용되는 다이아그램도 없지만 또 DFD 만큼 잘못되게 그려지는 다이아그램도 없다. 많은 개발자들이 정확한 DFD의 작성 방법을 몰라서, 또는 시간이 부족해서 부실한 다이아그램을 그리고 만다. DFD는 일반적으로 Context Diagram을 포함해서 레벨 3까지 분할되면 적당하며, 레벨간의 데이터흐름, 외부엔티티, 데이터저장소가 반드시 일치해야 한다. 또한 모든 데이터흐름과 데이터저장소는 프로세스에 연결되어 있어야 하고, 입력이나 출력 중 어느 하나만 있는 경우는 Blackhole과 Miracle이라 하여 옳지 않은 것으로 본다. 또한 데이터저장소간의 데이터흐름의 연결도 피해야 한다.  **6.1 데이터 보존의 원칙(Conservation Rule)**  어떤 프로세스의 출력 데이터흐름은 반드시 입력 데이터흐름을 이용하여 생성된 것이어야 한다는 것이다. 아래 예에서 사과라는 데이터는 쥬서라는 프로세스를 통해 사과 쥬스가 되어야 한다. 따라서 오렌지 쥬스라는 출력은 잘못된 것이다.    [그림 7] 데이터 보존의 원칙  **6.2 최소데이터 입력의 원칙**  최소데이터 입력의 원칙은 어떤 프로세스가 출력 데이터흐름을 산출하는데 반드시 필요로 하는 최소의 데이터흐름만을 입력해야 한다는 것이다. 아래 그림에서 보는 바와 같이 실급여액 계산이란 프로세스가 필요로 하지 않는 근무시간은 이 처리를 통과하지 않고 지나친다는 것을 보여주고 있다.    [그림 8] 최소데이터 입력의 원칙  **6.3 지속성의 원칙**  프로세스는 입력 데이터흐름이 들어오기만 한다면 항상 수행할 준비를 갖추고 있어야 한다는 것이다. 다음 그림에서 보는 한영번역 처리는 어떤 개인용 컴퓨터에서 실행할 수 있는 소프트웨어로 구매할 수도 있다. 이 때 번역 프로그램을 수행시키면 항상 번역을 실행할 준비를 하고 한글단어가 입력되기를 기다리고 있다는 점에서 지속성을 갖게 된다.    [그림 9] 지속성의 원칙  **6.4 순차처리의 원칙**  순차처리의 원칙은 데이터흐름 또는 데이터저장소로부터 입력되는 데이터에 대한 처리 순서를 설명하는 데, 데이터흐름을 통해 입력되는 데이터는 도착되는 순서대로 처리하는 반면, 데이터저장소로는 어떤 순서에 의해 접근하여도 무방하다는 것을 의미한다. 다음 그림에서 주목해야 할 것은 한영사전이란 프로세스에서 보는 것처럼 한글단어 데이터흐름의 첫번째 단어인 책이 맨 먼저 처리되어 book이란 단어로 데이터저장소에 저장되는데, 이들은 접근 순서에 무관하게 저장되어 있다.    [그림 10] 순차처리의 원칙  **6.5 영구성의 원칙**  영구성의 원칙은 데이터흐름의 데이터는 처리된 후 없어지지만 데이터저장소의 데이터는 아무리 읽어도 없어지지 않는다는 것을 말한다.  **6.6 데이터변환의 원칙**  분석시 불명확한 점의 하나는 어떤 종류의 프로세스를 데이터 흐름도 상에 나타내야 하는지 판단하기가 어렵다는 것이다. 이러한 경우에 도움이 될 수 있도록 4가지 종류의 프로세스 형태가 존재한다. 데이터본질의 변환, 데이터합성의 변환, 데이터관점의 변환, 데이터구성의 변환 등이 있다.  ▶ 데이터본질의 변환  아래 그림에서 보는 바와 같이 데이터흐름 본질의 변환은 일반적으로 입력 데이터흐름에 편집, 계산 등을 하여 출력 데이터흐름을 산출하는 것을 의미한다. 아래 그림에서 소득증가율 계산이라는 프로세스는 금액으로 표시된 소득의 값을 입력으로 받아 퍼센트로 표시된 증가율로 변환을 한다.    [그림 11] 데이터본질의 변환  ▶ 데이터합성의 변환  아래 그림에서 예금처리라는 프로세스는 입력데이터의 본질을 변화시키지 않는다. 이러한 형태의 프로세스는 단순히 하나의 입력데이터를 여러가지 구성요소로 분리하여 2개 이상의 출력을 산출한다. 반대로 2개 이상의 입력 데이터흐름에 대하여 데이터합성의 변환이 발생할 수도 있는데, 그림에서와 같이 수표와 입금표라는 데이터항목이 합성되어 입력 트랜잭션이라는 하나의 출력 데이터 흐름을 산출한다.    [그림 12] 데이터합성의 변환  ▶ 데이터관점의 변환  아래 그림에서 보는 것처럼 데이터관점의 변환은 데이터에 대해 실제적으로 변경을 하지는 않는다. 이러한 형태의 처리에서 입력 데이터 흐름은 동일하게 출력 데이터흐름으로 나타나게 된다. 그림에서 주문서 확인은 주문서에 대해 변경을 하지 않고 그대로 출력으로 내보내게 된다. 따라서 출력 데이터흐름의 이름을 적합한 주문서라 하였으며, 데이터사전에서도 주문서가 정의되어 있는 한 새롭게 정의할 필요가 없다.    [그림 13] 데이터관점의 변환  ▶ 데이터구성의 변환  데이터구성의 변환에서는 출력데이터가 입력데이터와 동일하지만, 데이터의 구성형태가 변환되게 된다. 구성의 변환은 포맷팅(Formatting) 또는 정렬(Sort) 등을 위한 처리를 필요로 하게 된다. 이러한 처리형태의 출력은 다른 데이터흐름으로 생각될 수도 있고 아닐 수도 있다. 다음 그림에서 판매보고서는 데이터사전에 새로운 데이터흐름으로 들어갈 수 있는데 보고서의 실물모형(Mock-Up)이 될 것이다. 만일 처리가 정렬을 위한 것이라면 출력을 입력과 다른 데이터흐름으로 간주하여 정렬된 판매데이터라 할 수 있다.    [그림 14] 데이터구성의 변환 |