〈研究論文〉

Journal of the Korean Institute of Industrial Engineers
Vol. 11, No. 2, Dec. 1985

Multi-Module 소프트웨어 시스템의 有用性 豫測†

(Estimation of Availability for Multi-Module Software Systems)

金 永 輝*
金 仲 煥**

Abstracts

This study deals with problems of estimating the availability of the multi-module software systems. The result presented in this paper is an extension of our previous paper (2) entitled "A modified Markov model for the estimation of computer software performance". The extension is made by assuming that (1) the software system consists of R statistically independent software modules; (2) no failure occurrence while the process is in transition between software modules.

1. 序 論

擴大一路에 있는 컴퓨터의 普及과 그 應用技術의 발달로 오늘날 우리사회의 모든 분야에서는컴퓨터의 역활이 날로 증대되어 가고 있다. 이는 곧 우리社會의 컴퓨터에 대한 依存度가 높아지고 있음을 의미하며 이러한 현상은 필연적으로 컴퓨터에 대한 높은 信賴性의 保障을 요구하는 결과를 초래하게 하였다.

과거 30여년간 micro electronics 분야의 눈부신 발달은 컴퓨터 hardware 기술의 비약적인 발전을 가져왔지만 software 기술의 발전은 hardware 에 비해 실로 미미한데 불과하였다. 또 software 가 구조적으로 hareware 에 비해 훨씬

* 高麗大學校 産業工學科
** 韓國外國語大學校 電算學科
† 本 研究는 韓國科學財團의 支援에 의하여 이루어진 것임.

더 복잡하기 때문에 컴퓨터 시스템의 信賴性을 保障하는 데에는 software system의 信賴性을 확보하는 것이 무엇보다도 중요하게 되었다.

소프트웨어 시스템의 높은 信賴性을 보장하기 위한 이상적인 試驗方法은 시스템에 부여될 수 있는 모든 가능한 入力들을 충분한 時間을 두고 實行하며 誤謬를 修正해 나가는 것이다. 그러나 現實的으로 자원(時間, 費用등)의 제한을 받게 되어 이러한 試驗方法은 사실상 불가능하다.

현실적으로 제한된 자원으로 요구되는 信賴性의 확보를 하는 과정에서 試驗을 위해 잘 알려진 入力들을 선택하며 이를 처리하는 적절한 實行時間을 결정하여 試驗하고 試驗過程에서 얻어진 자료를 分析하므로써 開發完了되는 시점에서 실제 운영시의 소프트웨어 信賴性(software reliability)을 豫測하는 일은 매우 중요하다.

소프트웨어 信賴性에 관한 尺度로는 信賴性, 有用性(availabilability), 시스템에 남아있는 誤謬의 갯수, 故障과 故障 사이의 時間등이 있다. 이러한 尺度중에서 실제 운영되고 있는 소프트웨어 시스템의 效率을 나타낼 수 있는 尺度는 有用性이라고 할 수 있다. 특히 온-라인 시스템과 같이 시스템에 부여된 入力을 신속 정확하게 처리해야 하고 保全(maintenance;修理 및 update) 가능한 시스템의 경우에는 有用性 豫測 模型 이 보다 적합하다.

소프트웨어 有用性 模型 으로는 Trivedi 와 Shooma- n(9) Okumoto 와 Goel(6), Kim et. al.(2), Shant- hikumar(8)의 模型등이 있다.

이와같은 模型에서는 모두 소프트웨어 시스템을 한개의 black-box 로 보아 시스템내의 모든 誤謬가 시스템의 故障발생에 균일한 기회로 영향을 미친다고 보았다.

그러나 시스템이 대형(10^5단위 이상의 단어)인 경우에 여러개의 독립된 module 들로 구성되며 각 module 들은 實行빈도의 차이가 있기 때문에 實行빈도가 적은 module 에 있는 誤謬보다는 實行빈도가 큰 module 에 있는 誤謬에 의하여 故障이 발생할 가능성이 크다.

그러므로 black-box 模型은 시스템의 開發을 각 module 별로 開發하고 이를 統合하는 경우에 統合段階에서 적용하기에 적합하다고 볼 수 있다.

이 硏究는 각 module 의 有用性을 豫測하면 유한개의 독립된 module 들로 구성된 multi-module 소프트웨어 시스템의 有用性도 豫測할 수 있는 경제적인 有用性 模型을 開發하는데 目的을 두고 있다. 이때 각 module 의 有用性은 필자에 의하여 開發된 Kim et. al. 模型의 硏究結果를 사용하여 豫測한다.

Littlewood(3) 의 연구에서는 소프트웨어 시스템에 入力이 주어지면 요구되는 처리조건에 따라 어느 한 module 이 實行되기 시작하여 주어진 入力의 처리가 끝날때까지 각 module 들이 實行되어지는 實行過程 (execution process)이 마코프過程(Markov process) 인 것을 전제로 하고 있다.

本 硏究에서도 소프트웨어 實行過程을 마코프過程으로 간주하고 模型을 設定하여 임의 時間에 각 module 들이 實行되고 있을 確率을 구하였다. 그리고 이를 각 module 의 有用性 豫測結果와 함께 multi-module 소프트웨어 有用性의 定義에 사용하므로써 각 module 의 實行빈도의 차이를 고려하여 시스템의 有用性을 豫測하였다.

2. 소프트웨어 實行過程의 마코프 模型

2.1實行過程

소프트웨어 시스템의 開發過程은 시스템의 복잡한 구조와 誤謬의 위치및 갯수등과 같은 특성을 파악하기 위하여 試驗단계를 중요시하는 관점에서 分析 및 設計, module 開發 및 試驗, 시스템 統合 및 기능 試驗, 保全단계로 區分한다. 이러한 단계를 거쳐 開發된 소프트웨어 시스템은 독립된 유한개의 module (예, FORTRAN 의 sub-routine)들로 구성되고 設計단계 및 시스템의 統合단계에서 각 module 사이의 상호연결되는 관계를 파악하여 이를 block-diagram으로 나타낼 수

있다.

시스템에 入力이 부여되면 요구되는 처리조건에 따라 어떤 한 module 이 實行되기 시작하여 block-diagram 의 경로를 따라 몇개의 module 이 實行되어지는 過程을 거쳐 작업을 완성하게 된다.

그림 2.1은 3개의 module로 구성된 가상적인 소프트웨어 시스템의 block-diagram 을 예시한 것이다.

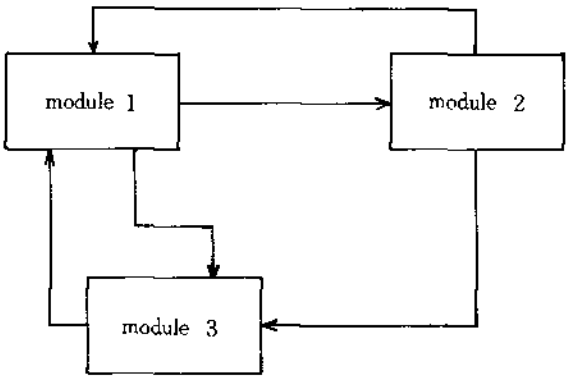

그림 2.1 가상적인 소프트웨어 시스템의 block-diagram

이때 각 module 의 實行에 소요되는 實行 時間은 주어진 入力을 처리하기 위하여 소요되는 task -processing 時間, dispatching 時間, TC-wait 時間(terminal control wait time), FC-wait 時間(file control wait time) 으로 구성된다. 그리고 시스템에 부여될 가능성이 있는 모든 入力들의 集合인 入力空間(input space), I 에서 임의 추출된 入力들이 시스템에 부여된다면 어떤 한 표본경로(sample path) 의 결과는 가상적인 시스템의 경우에 그림 2.2와 같이 예시할 수 있다.

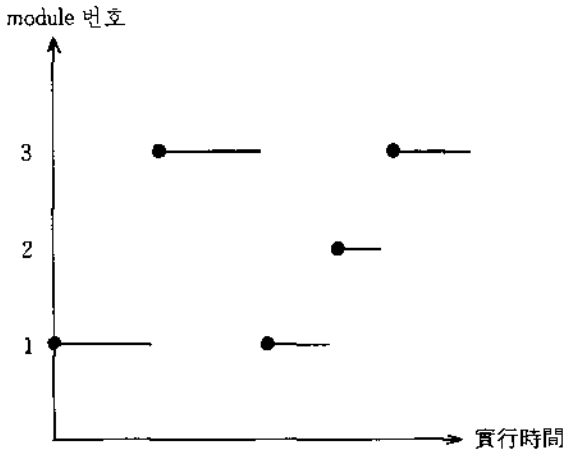

그림 2.2 임의의 한 표본경로의 결과

이러한 소프트웨어 實行過程의 behavior 는 그림 2.3과 같이 入力空間, I 의 임의의 한 入力a-와 出力空間(output space), O 의 한 出力b 를 對應시키는 函數, P : a→b 의 기능을 갖는 形態로 설명된다.

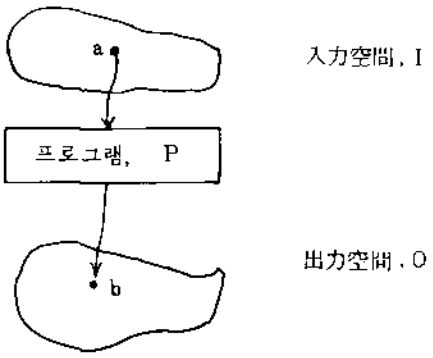

그림 2.3 프로그램의 기능

이와같은 기능을 갖는 소프트웨어 實行過程의 behavior 를 模型化하는 한 方法으로서는 시스템에 주어지는 入力들은 確率分布를 가지며 이러한 入力들이 주어지면 요구되는 처리조건에 따라 어떤 한 module 이 實行되기 시작하여 block-diagr-am 의 경로를 따라 몇개의 module 들이實行되어지는 實行過程을 마코프過程으로 假定하는 마코프 模型을 생각 할 수 있다.

소프트웨어 實行過程을 마코프 過程으로 假定한 것은 최초 Littlewood (3)의 模型에서 소프트웨어 시스템의 故障過程을 分析할 目的으로 사용되었다. 실제문제에서는 예를들면 FROTRAN의 subroutine 副프로그램을 호출하는 call 명령문과 같이 프로그램과 프로그램사이의 遷移는 現

在 實行되고 있는 프로그램에 의존한다는 것을 알 수 있다.

그러나 아직 실제 자료를 가지고 이러한 假定의 타당성에 대한 檢定 문제를 다룬 硏究가 없었으며 入力空間의 특성 및 시스템의 구조에 따라 소프트웨어 實行過程의 behavior 가 다르므로 임의의 소프트웨어 시스템의 實行過程을 마코프 過程으로 표현 할 수 있느냐 하는 것을 결정하는 것은 매우 어려운 문제이다.

2.2 模型의 設定 및 解

소프트웨어 시스템이 R개의 독립된 module 들로 구성되어 있고 그림 2.1의 예와 같은 block-diagram으로 각 module 사이의 관계를 나타낼 수 있으며 소프트웨어 實行過程은 마코프 過程이라고 하자. 그러면 소프트웨어 實行過程은 주어진 入力을 처리하기 위하여 주어진 상황이 요구하는 조건에 따라 어떤 특정한 module 이 實行되므로써 시작되고 그 module 이 實行完了되면 즉시 다른 module 이 實行되기 시작하여 주어진 入力의 처리가 끝날때 까지 각 module 들이 實行되어지는 過程이다. 이때 각 module 의 실행에 소요되는 實行時間은 task processing 時間, dispatch-ing 時間, FC-wait 時間으로 구성된다.

한편 시스템의 狀態空間(state space), E는아래와 같이 표시된다.

$$E=\{1, 2, 3, \cdots\cdots, R\} \qquad \langle 2.1\rangle$$

Y를 시간 t에 實行되고 있는 module번호라고 하면$\{Y_t=i\}$는 時間 t에 시스템내의 module i가 實行 되고 있는 狀態를 의미한다.

그러면 소프트웨어 시스템의 實行過程,

$$Y=\{Y_t : t\in[0, \infty)\} \qquad \langle 2.2\rangle$$

는 假定에 의하여 狀態空間, E를 갖는 마코프과정이다. 시점 T_0, T_1, T_2, ……는 實行過程의 狀態가 변하는 순간이며 Z_0, Z_1, Z_2, ……는 이들 시점에서의 Y의 狀態라고 하면 다음과 같은 결과를 얻을 수 있다.

$$T_0,\ T_{k+1}=T_k+W_{T_k},\ k\in N=\{0, 1, 2, \cdots\cdots\} \qquad \langle 2.3.1\rangle$$

$$Z_k=Y(T_k) \qquad \langle 2.3.2\rangle$$

단, W_{T_k} : T_k순간의 狀態에서 머무는 時間

여기서 $\{Z_k;k\in N\}$는 Markov chain이 되며 $T_{k+1}-T_k$는 모수가 Z_k에 의존하는 指數分布가된다. 그리고 實行過程이 time-homogeneous 마코프 過程일 때 Cinlar(1)의 pp. 247 定理(3·3) 에 의하여 $\{Z_k=i\}$ 일 때 $i, j\in E$ 및 $t\in R_+=[0, \infty]$에 대하여

$$P\{Z_{k+1}=j, T_{k+1}-T_k\rangle t/Z_0, \cdots\cdots, Z_k;\ T_0, \cdots\cdots, T_k\}=q_{ij}\exp(-\phi_i t) \qquad \langle 2.4\rangle$$

단, $q_{ij}\geq 0$, $q_{ii}=0$, $\sum_j q_{ij}=1$

$q_{ij}=P\{Z_{k+1}=j/Z_k=i\}$

ϕ_i;module i가 實行되고 있는 狀態에서의 固定된 遷移率(constant transitionn rate)이 된다.

狀態空間, E가 유한集合이므로 마코프 過程 Y는 regular 마코프 過程이며

$$P\{Y_t=i\}=g_i(t) \qquad \langle 2.5\rangle$$

라고 하면 Kolmogorv's forward equations은

$$\dot{g}_i(t)=-\phi_i g_i(t)+O\sum_{j=1, j\neq i}^{R}\phi_i q_{ij} g_j(t) \qquad \langle 2.6\rangle$$

$i=1, 2, \cdots\cdots, R$

단, $\dot{g}_i(t)=dg_i(t)/dt$

이며 〈2·6〉식을 行列을 사용해서 표시하면 generatorA(i, j)가

$$A(i, j)=\begin{cases}-\phi_i & ;i=j\\ \phi_i q_{ij} & ;i\neq j\end{cases} \qquad \langle 2.7\rangle$$

이므로 다음과 같이된다.

$$\dot{G}(t)=G(t)\cdot A \qquad \langle 2\cdot 8\rangle$$

단, $A = A(i, j)$

$G(t) = [g_1(t), g_2(t), \cdots\cdots, g_R(t)]$

$\dot{G}(t) = [\dot{g}_1(t), \dot{g}_2(t), \cdots\cdots, \dot{g}_R(t)]$

초기조건, $G(0) = [g_1(0), g_2(0), \cdots, g_R(0)]$

따라서 〈2.8〉식의 解는 다음과 같이 된다.

$$G(t) = G(0) \cdot \exp(At)$$

3. 소프트웨어 有用性 豫測

3.1 한개 module 으로 이루어진 시스템의 경우

소프트웨어 module 은 초기시간 t = 0에 module 내에 未知갯수 n 개의 誤謬을 포함한 狀態에 최초의 入力을 처리하기 위하여 實行되며 實行중 故障이 발생되면 故障원인이 되는 誤謬를 修正하게 된다.

이러한 實行過程에서 故障의 발생없이 實行되고 있는 狀態를 module 의 稼動狀態(up-state)라고 하며 誤謬를 修正하고 있는 狀態를 module의 故障狀態(down-state) 라고 하자. 그러면 module 은 module 내의 모든 誤謬가 完全히 修正될때까지 稼動狀態와 故障狀態를 반복하면서 實行되어 지며 모든 誤謬가 完全히 修正되면 항상 稼動狀態에 머무르게 된다.

X_t 를 시간 t 에 module 내에 있는 誤謬의 갯수라고 한다면 $\{X_t = k\}$ 는 시간 t 에 module내에 k 개의 誤謬가 있는 狀態를 의미한다. 그러므로 $P\{Xt = k\}$ 는 시간 t 에 module 내에 k 개의 誤謬가 있을 確率이다.

한편 故障이 일어나게 되면 故障原因이 되는 誤謬를 제거하기 위하여 프로그램을 修正하게 되는데 그 結果로서는 다음 두가지 경우를 생각할 수 있다. 즉 故障이 원인이 되는 誤謬를 正確히 발견하여 올바른 시정조치를 취함으로서 誤謬 한 개를 제거하는 경우와 修正作業이 미숙하여 誤謬 하나를 제거했지만 그 대신 또 다른 誤謬를 범하여 結果的으로 프로그램에 포함되어 있는 총 갯수는 변하지 않는 경우이다.

물론 프로그램을 修正하는 과정에서 誤謬를 2 개이상 범하게 되어 結果的으로 誤謬의 총 갯수가 증가하는 경우도 생각할 수 있으나 여기서는 그 발생 可能性이 미미하다는 점을 고려하여 이를 무시하는 것으로 한다. 지금 소프트웨어 module 의 有用性 模型의 設定을 위해 다음과 같은 假定을 한다고 하자.

1) 稼動중에 있는 소프트웨어 module 의 故障率은 module 내에 포함되어 있는 誤謬의 갯수에 비례한다. 즉 임의의 시간 t에 $X_t = k$라 할 때 이 module 의 故障率은 $k\lambda$ 이다. 단, k는 상수.

2) 일단 소프트웨어 module 에 故障이 발생하여 프로그램을 修正하는 과정에서 올바른 修正을 하게되어 module 내에 포함된 誤謬의 갯수가 한개 감소하여 module 이 稼動狀態로 들어갈 때의 修理率은 μ_0 이와 반대로 修正하는 과정에서 또 다른 誤謬를 범하여 module 내의 誤謬갯수가 변하지 않은 狀態에서 module 이 稼動狀態로 들어갈 때의 修理率은 μ_1 으로 한다. 단, μ_0 와 μ_1은 誤謬의 수에 관계 없이 상수인 것으로 한다.

표시를 간략하기 위하여 時間 t 에서 module내에 포함된 誤謬의 갯수를 k, 즉 $Xt = k$ 일 때 module 이 稼動狀態에 있을 確率을 $P_k(t)$, module - 이 故障狀態에 있을 確率을 $q_k(t)$로 하자. 그러면 module 의 초기, 즉 t = 0 일때 module에 포함된 誤謬의 갯수를 n이라 한다면 $P_k(t)$, $q_k(t)$ 사이에는 다음과 같은 관계식이 성립한다. 즉

$$\sum_{k=0}^{n} P_k(t) + \sum_{k=1}^{n} q_k(t) = 1 \qquad \langle 3.1 \rangle$$

그리고 $P_k(t)$와 $q_k(t)$에 관한 微分方程式은 다음과 같다.

$$\dot{P}_n(t) = -n\lambda P_n(t) + \mu_1 q_n(t) \qquad \langle 3.2.1 \rangle$$

$$\dot{P}_{n-k}(t) = -(n-k)\lambda P_{n-k}(t) + \mu_0 q_{n-k+1}(t) + \mu_0 q_{n-k}(t)_{n-k} \qquad \langle 3.2.2 \rangle$$

$$\dot{P}_0(t) = \mu_0 q_1(t), \; k = 1, 2, \cdots\cdots, n-1 \qquad \langle 3.2.3 \rangle$$

$$\dot{q}_{n-k}(t) = -(\mu_0+\mu_1)\,q_{n-k}(t) + (n-k)\,\lambda P_{n-k}(t)$$
$$, k=0, 1, \cdots\cdots, n-1 \qquad \langle 3.3\rangle$$

초기조건 : $P_n(0)=1, \dot{P}_n(0)=-n\lambda$ $\langle 3.4.1\rangle$

$$P_{n-k}(0)=0, \dot{P}_{n-k}(0)=0 \qquad \langle 3.4.2\rangle$$
$$, k=1, 2, \cdots\cdots, n$$

$$q_{n-k}(0)=0, k=0, 1, \cdots, n-1 \qquad \langle 3.4.3\rangle$$

〈3.2〉식을 t에 관하여 微分하여 〈3.3〉식에 대입하여 정리하면 다음과 같은 2계微分方程式이 된다.

$$\ddot{P}_n(t)+[\mu_0+\mu_1+n\lambda]\dot{P}_n(t)+n\lambda\mu_0 P\ P_n(t)=0$$
$\langle 3.5.1\rangle$

$$\ddot{P}_{n-1}(t)+[\mu_0+\mu_1+(n-k)\lambda]\dot{P}_{n-k}(t)+(n-k)$$
$$\lambda\mu_0 P_{n-k}(t)=(n-k+1)\lambda\mu_0 P_{n-k+1}(t) \langle 3.5.2\rangle$$
$$k=1, 2, \cdots, n$$

단, $\ddot{P}_k(t)=d^2P_k(t)/dt^2$

$$P_n(0)=1, \dot{P}_n(0)=-n\lambda \qquad \langle 3.6.1\rangle$$
$$P_{n-k}(0)=0, \dot{P}_{n-k}(0)=0 \qquad \langle 3.6.2\rangle$$
$$, k=1, 2, \cdots, n$$

Laplace 變更을 사용하여 이 微分方程式의 解를 구하면 같이 된다.

$$P_n(t)=\frac{A(n)+\mu_0+\mu_1}{A(n)-B(n)}\exp[A(n)t]$$
$$+\frac{B(n)+\mu_0+\mu_1}{B(n)-A(n)}\exp[B(n)t \qquad \langle 3.7.1\rangle$$

$$P_{n-k}(t)=(\lambda\mu_0)^k\binom{n}{k}k!\sum_{j=0}^{k}\left\{\frac{A(n-j)+\mu_0+\mu_1}{C_A(i,j)}\right.$$
$$\exp[A(n-j)t]+\frac{B(n-j)+\mu_0+\mu_1}{C_B(i,j)}$$
$$\left.\exp[B(n-j)t]\right\}, k=1, 2, \cdots, n \qquad \langle 3.7.2\rangle$$

단, $A(n-k)\neq B(n-k)$

$$\begin{matrix}A(n-k)\\B(n-k)\end{matrix}\Big\} =\frac{1}{2}\{-[\mu_0+\mu_1+(n-k)\lambda]\pm$$
$$\sqrt{[\mu_0+\mu_1+(n-k)\lambda]^2-4(n-k)\lambda\mu_0}$$
$\langle 3.7.3\rangle$

$$C_A(i,j)=\Pi_{i=0,\cdots,j}^{*}[A(n-j)-A(n-i)]$$
$$\Pi_{i=0}^{*}[A(n-j)-B(n-i)] \qquad \langle 3.7.4\rangle$$

$$C_B(i,j)=\Pi_{i=0,\cdots,j}^{*}[B(n-j)-B(n-i)]$$
$$\Pi_{i=0}^{*}[B(n-j)-A(n-i)] \qquad \langle 3.7.5\rangle$$

3.2 Multi-Module 시스템의 경우

소프트 웨어 시스템이 獨立된 R개의 module 소프트웨어 시스템이며 시스템에 주어진 入力을 처리하기 위하여 주어진 처리조건에 따라 각 module 들이 實行되는 實行過程이 마코프過程 이라고 하자. 그리고 시스템의 故障은 module 사이를 遷移할 때는 발생되지 않으며 각 module 의 實行중에만 발생한다고 하자.

이와같은 경우에 각 module들이 최초에 포함하고 있는 誤謬의 갯수를 n_i (i=1, 2, ……, R) 개라 하고 1개의 誤謬에 의한 故障 發生率을 λ_i (i=1, 2, ⋯, R) 라고 한다면 model i의 故障率은 $n_i\lambda_i$가 된다. 또한 module i에서 故障이 발생하여 誤謬 1개가 감소한 狀態에서 稼動狀態로 갈때의 修理率을 μ_{i0}라고 하며 이와 반대로 修理作業이 미숙하여 誤謬 1개를 제거 하였지만 또 다른 誤謬를 범하여, module 내의 誤謬갯수가 변하지 않은 狀態에서 稼動狀態로 갈때의 修理率을 μ_{i1}이라고 하자. 그리고 multi--module 소프트웨어 시스템의 有用性, $A_s(t)$를 다음과같이 定義할 수 있다.

$$A_s(t)=\sum_{i=1}^{R}\sum_{\ell=0}^{n_i} P\{\text{module이 } \ell\text{개의 誤謬을 가진채 稼動/시간 t에 module i가 實行되고 있는 狀態}\}\cdot P\{\text{時間 t에 module i가 實行되고 있는 狀態}\} \qquad \langle 2.8\rangle$$

이 定義에서 사상(event), {時間 t에 module i

가 實行되고 있는 狀態}는 {$Y_t=i$}이며 사상 {module이 ℓ개의 誤謬를 가진채 稼動狀態/時間 t에 model i가 實行되고 있는 狀態}는 {$X_t=\ell$ / $Y_t=i$}이다.

〈2.5〉식에 의해 $P\{Y_t=i\}=g_i(t)$이며 〈2.5〉식과 $P\{X_t=k\}=P_k(t)$에 의하여

$$P\{X_t=\ell/Y_t=i\}=P_{\ell i}(t) \qquad \langle 3.9\rangle$$

로 표시하면 〈4.8〉식은 다음과 같이 된다.

$$A_s(t)=\sum_{i=1}^{R}\sum_{\ell=0}^{n_i}P\{X_t=\ell/Yt=i\}\cdot P\{Y_t=i\}$$

$$=\sum_{i=1}^{R}\sum_{\ell=0}^{n_i}P_{\ell,i}(t)\cdot g_i(t) \qquad \langle 3.10\rangle$$

여기서 $g_i(t)$는 〈2·9〉식에 의하여 $P_{\ell,i}(t)$는 〈3.7〉식에 의해

$$P_{\ell,n_i}(t)=\frac{A(n_i)+\mu_{i0}+\mu_{i1}}{A(n_i)-B(n_i)}\exp[A(n_i)t]$$

$$+\frac{B(n_i)+\mu_{i0}+\mu_{i1}}{B(n_i)-A(n_i)}\exp[B(n_i)t] \qquad \langle 3.11,1\rangle$$

$$P_{i,n_i-k}(t)=(\lambda_i\mu_{i0})^k\binom{n_i}{n}k!$$

$$\sum_{j=0}^{k}\left\{\frac{A(n_i-j)+\mu_{i0}+\mu_{i1}}{C_A(m,j,n_i)}\exp[A(n_i-j)t]\right.$$

$$\left.+\frac{B(n_i-j)+\mu_{i0}+\mu_{i1}}{C_B(m,j,n_i)}\exp[B(n_i-j)t]\right.$$

$$,k=1,2,\cdots,n \qquad \langle 3,11,2\rangle$$

단, $A(n_i-k)\neq B(n_i-k)$

$$\left.\begin{matrix}A(n_i-k)\\B(n_i-k)\end{matrix}\right\}=\frac{1}{2}\cdot[\mu_{i0}+\mu_{i1}+(n_i-k)\lambda_i]\pm$$

$$\sqrt{[\mu_{i0}+\mu_{i1}+(n_i-k)\lambda_i]^2-4(n_i-k)\lambda_i\mu_{i0}}\}$$

$$C_A(m,j,n_i)=\Pi_{m=0}^{k}[A(n_i-j)-B(n_i-m)]$$

$$\Pi_{m=0,\,m\neq j}^{k}[A(n_i-j)-A(n_i-m)]$$

$$C_B(m,j,n_i)=\Pi_{m=0}^{k}[B(n_i-j)-A(n_i-m)]$$

$$\Pi_{m=0,\,m\neq j}^{k}[B(n_i-j)-B(n_i-j)]$$

이 된다.

그러므로 multi-module 소프트웨어 시스템의 有用性, $A_s(t)$는 〈3.11〉식과 $g_i(t)$를 〈3.10〉 식에 대입하면

$$A_s(t)=\sum_{i=1}^{R}\sum_{k=0}^{n_i}P_{i,n_i-k}(t)\cdot g_i(t) \qquad \langle 3.12\rangle$$

와 같이 된다.

4. 模型 適用의 例

模型 適用의 可能性과 適用方法을 보이기 위하여 그림 4.1과 같은 가상적인 소프트웨어 시스템에 대하여 本 模型을 적용하여 보겠다.

그림 4.1과 같이 3개의 module로 구성된 소프트웨어 시스템에서 각 module의 誤謬 갯수, 故障率, 修理率, 平均 實行時間 및 각 module 사이의 遷移確率은 이미 그림 4.1에 주어진 것과 같이 推定되었다고 하자.

이때 시스템에 최초로 入力이 주어져서 module 1부터 實行되기 시작 하였다면 $P\{Yt=i\}=g_i(t)$ 의 값은 〈2.9〉식에 의하여 표 4.1과 같이 주어진다. 그리고 각 module의 有用性은 표 4.2와 같다. 또 시스템의 有用性은 표 4.3과 같이 豫測되며 그림 4.2와 같은 그래프가 된다.

한편 시스템을 하나의 black-box로 보아 각 module의 故障率의 加重平均을 시스템의 故障率로 간주한 black-box 模型의 경우와 本 模型의 경우를 비교한 그래프는 그림 4.3에 주어져 있다.

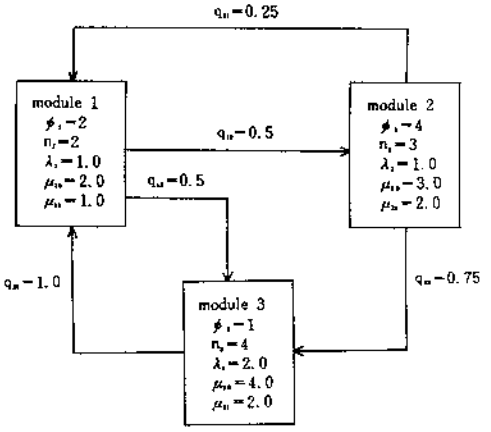

그림 4.1 가상적인 시스템의 block-diagram

표 4 . 1 가상적인 시스템의 $g_i(t)$ 의 값

g(t) t	0.1	0.3	0.5	1.0	1.5	2.0	2.5	3.0
$g_1(t)$	0.8272	0.6044	0.4821	0.3665	0.3407	0.3350	0.3337	0.3334
$g_2(t)$	0.0745	0.1285	0.1306	0.1028	0.0889	0.0847	0.0837	0.0834
$g_3(t)$	0.0983	0.2671	0.3873	0.5307	0.5703	0.5803	0.5826	0.5832

표 4.2.1 module 1 의 有用性

$P_{n-k}(t)$ t	0.1	0.3	0.5	1.0	1.5	2.0	2.5	3.0
$P_6(t)$	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
$P_5(t)$	0.038	0.148	0.193	0.162	0.102	0.059	0.032	0.017
$P_4(t)$	0.000	0.012	0.045	0.135	0.160	0.137	0.101	0.068
$P_3(t)$	0.000	0.000	0.003	0.041	0.106	0.151	0.162	0.147
$P_2(t)$	0.000	0.000	0.000	0.005	0.032	0.082	0.137	0.176
$P_1(t)$	0.000	0.000	0.000	0.000	0.004	0.021	0.057	0.109
$P_0(t)$	0.000	0.000	0.000	0.000	0.000	0.002	0.009	0.027
A (t)	0.038	0.160	0.241	0.343	0.403	0.452	0.498	0.544

표 4.2.2 modele 2 의 有用性

$P_{n-k}(t)$ t	0.1	0.3	0.5	1.0	1.5	2.0	2.5	3.0
$p_3(t)$	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
$p_2(t)$	0.033	0.163	0.267	0.351	0.309	0.236	0.168	0.114
$p_1(t)$	0.000	0.007	0.033	0.156	0.281	0.355	0.378	0.364
$p_0(t)$	0.000	0.000	0.001	0.017	0.068	0.154	0.260	0.372
A (t)	0.033	0.170	0.301	0.524	0.658	0.745	0.806	0.850

표 4.2.3 module 3 의 有用性

$p_{n-k}(t)$ t	0.1	0.3	0.5	1.0	1.5	2.0	2.5	3.0
$p_4(t)$	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
$p_3(t)$	0.084	0.241	0.250	0.129	0.050	0.017	0.006	0.002
$p_2(t)$	0.002	0.048	0.136	0.242	0.184	0.104	0.050	0.023
$p_1(t)$	0.000	0.003	0.022	0.166	0.292	0.305	0.249	0.177
$p_0(t)$	0.000	0.000	0.001	0.035	0.162	0.355	0.550	0.708
A (t)	0.086	0.292	0.409	0.572	0.687	0.781	0.856	0.909

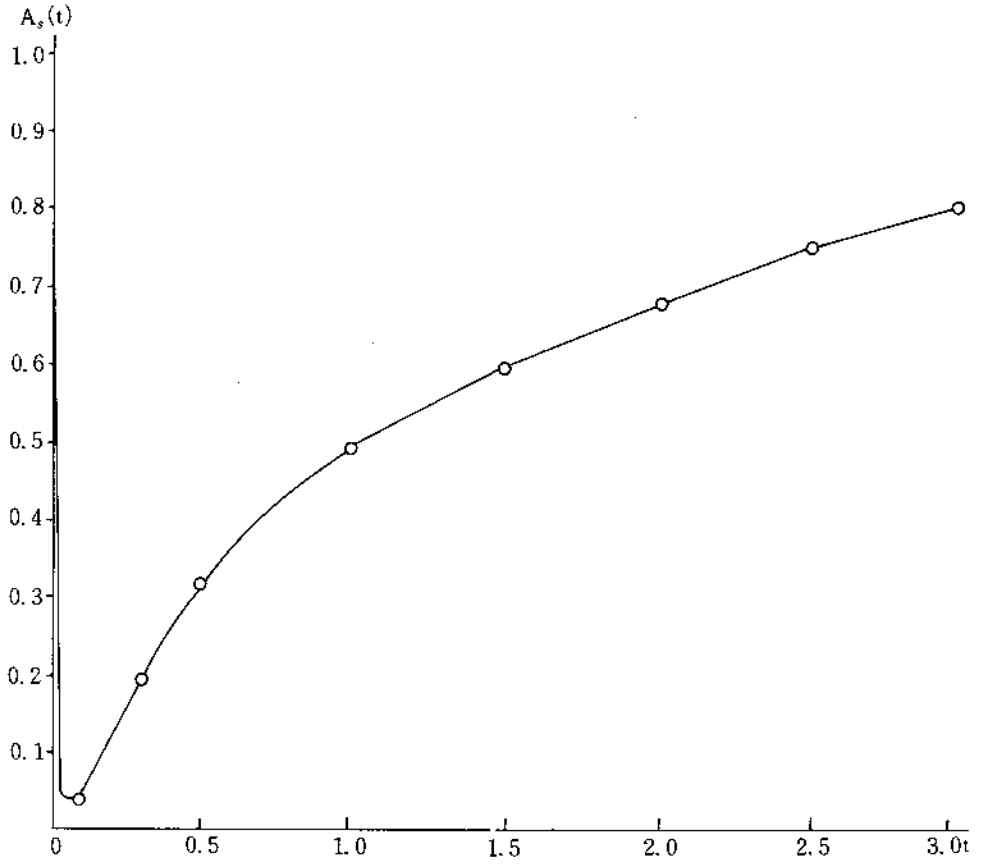

그림 4.2 시스템의 有用性 그래프

표 4.3 시스템의 有用性

$A_i(t)$ t	0.1	0.3	0.5	1.0	1.5	2.0	2.5	3.0
$A_1(t)$	0.032	0.097	0.116	0.126	0.137	0.152	0.166	0.181
$A_2(t)$	0.002	0.022	0.039	0.054	0.058	0.063	0.067	0.071
$A_3(t)$	0.008	0.078	0.158	0.304	0.392	0.453	0.499	0.530
$A_s(t)$	0.043	0.197	0.314	0.483	0.588	0.668	0.732	0.783

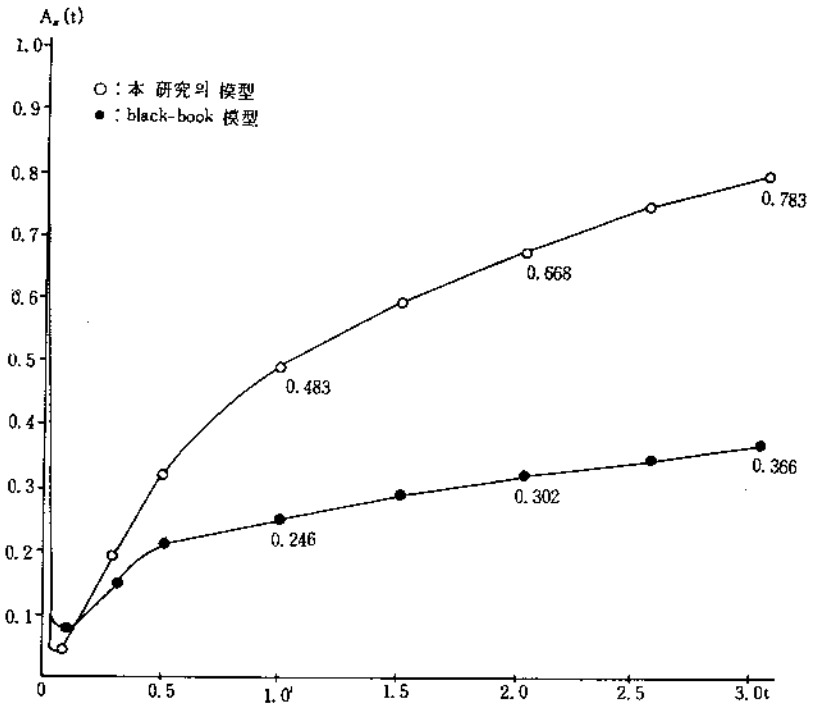

그림 4.3 加重平均에 의한 模型과 本 模型의 시스템의 有用性 비교그래프(가상적인 시스템의 경우)

5. 結 論

Rizza와 Hacker(7)는 컴퓨터 소프트웨어 시스템의 開發을 각 module별로 開發하고 이를 統合하는 경우에 統合段階에서의 試驗費用은 각 module 開發段階에서의 試驗費用보다 약 3배 가량 많이 소요된다고 하였다.

이러한 면에서 볼 때 本 研究에서 開發된 模型은 각 module의 有用性이 豫測되고 각 module사이의 관계가 把握될 때 multi-module 소프트 웨어 시스템의 有用性도 豫測할 수 있는 매우 경제적인 模型이라고 할 수 있다.

本 模型의 適用結果는 그림 4.3의 가상적인 시스템에 의하여 얻어진 그래프에 나타난 것과 같이 기존 black-box 模型과 비교하여 볼 때 實行빈도가 큰 module에 있는 誤謬는 實行 빈도가 적은 module의 誤謬보다 상대적으로 빨리 제거될 可能性이 크다는 점이 고려 되므로써 시스템의 사용초기에 有用性의 增加率이 크다는 Littlewood (4, 5)의 주장과 일치하였다.

이미 언급한 것과 같이 소프트웨어 시스템의 實行過程은 人力空間의 특징과 시스템 構成形態에 따라 behavior가 달라지게 되며 초시시간에 어느 module 부터 實行되느냐에 따라 다소 有用性 增加形態가 변할 수 있다. 따라서 초기 시간에 어느 module부터 실행되느냐 하는 것을 人力의 分布와 연관시키는 研究도 可能할 것으로 생각 된다.

本 研究에서는 結果들을 이용하여 계획된 소프트웨어 시스템의 有用性이 되기 위한 實行期間에 관한 問題와 각 module의 誤謬갯수, 故障率 및 修理率과 같은 母數를 推定하는 問題는 (3.7)식과 같은 解析的인 解를 사용하여 해결할 수 있다고 생각되며 계속 研究되어야 할 問題로 보여진다.

Reference

1. E. Cinlar, "Introduction to Sochastic Processes," Prentice-Hall, Inc., Englewood Cliffs, N.J. (1975).
2. J.H.Kim, Y.H.Kim, and C.J. Park, "A Modified Markov Model for the Estimation of Computer Software Performance," J. of O.R. Letters, Vol. 1-6 (Dec. 1982), pp. 253-257.
3. B. Littlewood, "A Reliability Model for Systems with Markov Structure," Applied Statistics CJ. Royal Statist. Soc. Series C Vol. 24, (1975), pp. 172-177.
4. B. Littlewood, "How to Measure Software Reliability and How not to...," Proc. 3rd. Intern. Conf. Software Engineering, Atlanta, Georgia (May, 1978), pp. 37-45.
5. B. Littlewood, "Theories of Software Reliability:How Good are They and How can They be Improved?" IEEE Trans. on Software Eng. Vol. SE-6, NO. 5 (Sept. 1980), pp. 489-500.
6. K. Okumoto and A.L. Goel, "Availability and Other Performance Measures of System under Imperfect Maintenance," Proc. COMPSAC (1978), pp. 66-71.
7. J.B. Rizza and D. Hacker, "Quality Assurance Inspection and Test Tools-An Application," Proc. of a Workshop on Currently Available Program Testing Tools, (1975), pp. 9-10.
8. J.G. Shanthikumar, "On a Software Availability Model with Imperfect Maintenance," Working Paper 83-011, Univ. of Arizona (Apr. 1983).
9. A.K. Trivedi and M.L. Shooman, "A Many State Markov Model for the Estimation and Prediction of Computer Software Performance Parameters," Proc. International Conference on Reliable Software (1975), pp. 208-220.