УДК 658.012.011:656.2.001.57+06 Дата подачи статьи: 09.09.15
DOI: 10.15827/0236-235X.112.158-165

ОЦЕНКА НАДЕЖНОСТИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ МЕТОДАМИ ДИСКРЕТНО-СОБЫТИЙНОГО МОДЕЛИРОВАНИЯ

(Работа выполнена при финансовой поддержке РФФИ, проекты №№ 15-08-01886-а, 15-01-03067-а)

М.А. Бутакова, д.т.н., профессор, декан, butakova@rgups.ru;
А.Н. Гуда, д.т.н., профессор, проректор, guda@rgups.ru;
А.В. Чернов, д.т.н., профессор, зав. кафедрой, avche@yandex.ru;
С.В. Чубейко, к.т.н., greyc@mail.ru
(Ростовский государственный университет путей сообщения,
пл. им. Ростовского Стрелкового полка Народного ополчения, 2, г. Ростов-на-Дону, 344038, Россия)

В статье рассматривается дискретно-событийное моделирование и представлены его отличительные особенности от других видов моделирования. Основное отличие – отсутствие привязки ко времени: достаточно соблюдать последовательность наступления событий, при этом не важно, какой временной промежуток будет между событиями. Дано определение модели дискретно-событийной системы с дополнением ее модельными часами, которые воспроизводят хронологию событий. Решается важная задача генерации списка событий различными способами: объектно-ориентированное и процессно-ориентированное исполнение событий. Подробно рассматриваются оба способа: приводятся иллюстрация, алгоритм и элемент программной реализации. События могут объединяться в группы, которые называются процессами. Процессно-ориентированное моделирование сложнее объектно-ориентированного, так как имеется планировщик процессов. Также в статье рассматривается оценка надежности ПО, базирующаяся на дискретно-событийном подходе. Данный подход основан на идее роста надежности ПО. Поиск ошибок моделируется случайным точечным процессом. При обнаружении ошибки она устраняется, тем самым ПО становится более надежным. Моделирование делится на две части: генерация процессов, имитирующих появление ошибок в ПО, и оценка системной надежности компонентного ПО. В статье рассматриваются варианты расчета вероятности возникновения ошибки в зависимости от структуры программ: последовательная, разветвляющаяся, циклическая и параллельная структура программного компонента. Для каждого варианта представлен иллюстрирующий рисунок и приведена вычислительная схема. Для циклической схемы программного компонента используется вычислительная схема последовательного компонента, так как это своего рода однотипные повторы последовательной структуры программного компонента.

Ключевые слова: *дискретно-событийное моделирование, событийно-ориентированное моделирование, процессно-ориентированное моделирование, надежность программного обеспечения, структура программного компонента.*

В данной работе рассматриваются аспекты моделирования широкого класса систем, причем именно признаки событийности являются наиболее существенными для адекватного составления математических моделей. Признаки событийности и построенные на них методы моделирования [1–3] существенно отличают исследуемый подход от методов моделирования, общепринятых, например, в теории систем и теории автоматического управления, основными инструментами которых являются интегро-дифференциальные уравнения, методы теории вероятностей, математической статистики и случайных процессов. В теории систем и теории автоматического управления обычным описанием исследуемой системы является описание вход-выход, а изменение выхода относительно входа, то есть пространство состояний системы, задается передаточными функциями в матричном виде. Большинство систем, являющихся объектами моделирования, нелинейные, и в данном случае существенные усилия моделирования направлены на линеаризацию систем, выполняемую путем решения систем дифференциальных уравнений в матричном виде. Результатами моделирования являются восстановленное фазовое пространство моделируемых систем и характеристики звеньев управления и регулирования.

Другим подходом является имитационное моделирование [4], использующее методы теорий *систем* и *сетей массового обслуживания* (СМО и СеМО соответственно) [5, 6], в которых рассматриваются различные модели входных, выходных потоков и правил обслуживания, построенных на базе соответствующих законов распределения случайных величин и процессов. В данном случае методами моделирования являются генераторы некоторых типов случайных процессов (имитирующих моменты времени поступления заявок на обслуживание), а результатами моделирования (часто усредненными) – времена пребывания заявки в очереди, системе, времена обслуживания, вероятности пребывания в очереди, вероятности обслуживания (за некоторый период) и другие вероятностные и статистические характеристики.

Заметим, что и в общей теории систем, и в теориях СМО и СеМО неявно предполагается использование процессного времени. Это означает, что в первом случае принимается, что процессы в системах протекают по возможности мгновенно (хотя в некоторых случаях допустима их инерционность), а во втором случае принимается, что процессы подчинены некоторому вероятностному закону. Однако во втором случае иногда рассматривают потоки *general* (общего) типа, в которых основным

соотношением является рекуррентное уравнение Линдли [7], что по сути близко к рассматриваемому далее подходу с идейной стороны, но не со стороны реализации методов моделирования. В целом процессное время означает, что изменение состояний системы, а также ее модели можно отметить на некоторой временной шкале; если шкала является непрерывной, то естественным будет непрерывное моделирование состояний системы, иначе – дискретное, а также соответствующий им математический аппарат.

В основу дискретно-событийного моделирования, развиваемого появления известной системы GPSS [8] и сетей Петри [9, 10] положена другая концепция: состояния системы изменяются под воздействием некоторых событий, в общем случае безотносительно к их точной привязке к временной шкале. Существенными являются лишь факты возникновения этих событий и взаимодействие их между собой, то есть синхронизация (некоторое событие предшествует другому, некоторое событие вызывает возникновение другого либо других событий и так далее). Примером таких информационных систем являются сетевые компьютерные системы. Для сетевых компьютерных систем как многопользовательских многозадачных, многомашинных, многопроцессорных систем характерным является еще один аспект событийности – конкуренция за сетевые распределенные вычислительные ресурсы с целью увеличения производительности, минимизации простоев и тому подобное. Оба этих аспекта (синхронизация и конкуренция) делают сетевые компьютерные системы существенно нелинейными, что усложняет их аналитическое и имитационное моделирование, а рассматриваемый далее в работе подход можно воспринимать как возможную линеаризацию таких систем.

Подходы и алгоритмы дискретно-событийного моделирования

Проведем грань между дискретно-событийным моделированием и другими методами моделирования более четко. Как уже упоминалось, большинство систем моделируется по принципу «вход–состояние–выход». Принимая общеизвестные обозначения векторов: $\mathbf{u}(t)$ – вход, $\mathbf{x}(t)$ – состояние, $\mathbf{y}(t)$ – выход, динамика моделируемой системы описывается уравнениями

$$\mathbf{x}'(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), t), \quad (1)$$

$$\mathbf{y}(t) = \mathbf{g}(\mathbf{x}(t), \mathbf{u}(t), t) \quad (2)$$

с начальными условиями $t > 0$.

Уравнение (1) означает составление множества состояний моделируемой системы, а если принять, что множество таких состояний равно n, а множество входных сигналов равно m, то получается, что необходимо моделировать n уравнений состояний:

$$x_1'(t) = f_1(x_1(t),...,x_n(t),u_1(t),...,u_m(t),t),$$

…,

$$x_n'(t) = f_n(x_1(t),...,x_n(t),u_1(t),...,u_m(t),t)$$

и k выходных уравнений системы:

$$y_1(t) = g_1(x_1(t),...,x_n(t),u_1(t),...,u_m(t),t),$$

...,

$$y_k(t) = g_k(x_1(t),...,x_n(t),u_1(t),...,u_m(t),t)$$

с начальными условиями $x_1(t_0) = x_1, \ldots, x_n(t_0) = x_n$.

В зависимости от вида функций f и g, способа фиксации моментов времени t в (1, 2) осуществляется непрерывное (или дискретное) моделирование нелинейной (или линейной) динамической системы.

Еще раз следует заметить, что изменение состояний динамической системы в таких случаях моделирования всегда привязано ко времени, какими бы его измерениями мы ни пользовались, непрерывными или дискретными. В дискретно-событийном моделировании важен факт фиксации события (или группы событий), а в какое время, либо интервал времени, либо через какой промежуток времени эти события фиксируются, уже не столь важно. Получается, что «событие первично, а время вторично». В работе [11] имеются иллюстрации к вышесказанным положениям, которые приведены в доработанном виде на рисунке 1. В верхней части

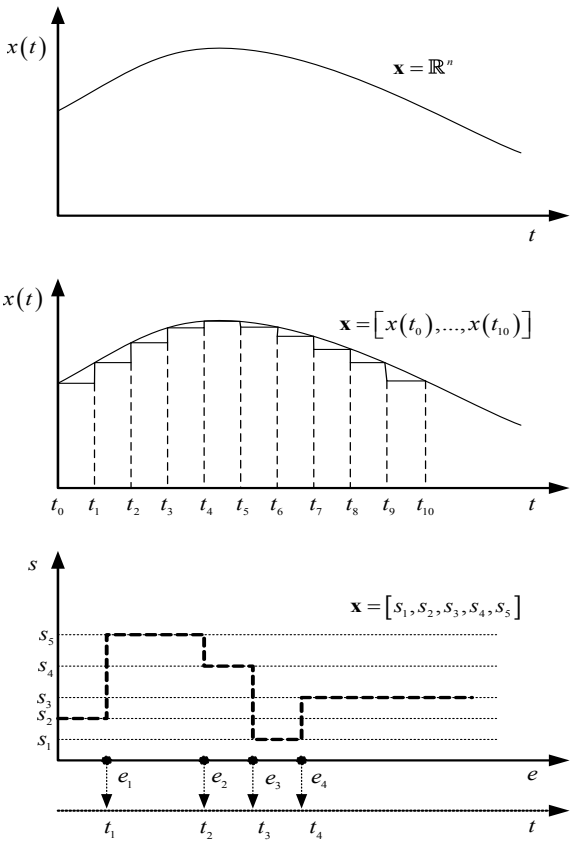

Рис. 1. Непрерывная, дискретная и дискретно-событийная системы

Fig. 1. Continuous, discrete and discrete-event systems

показано пространство состояний непрерывной системы, в средней – дискретной системы, а в нижней – дискретно-событийной системы. Ясно, что пространство состояний дискретно-событийной системы является дискретным и составляет события $s_1, \ldots, s_5$, а переключение между этими состояниями происходит в соответствии с наступлением некоторых событий $e_1, \ldots, e_4$. Естественно, при динамической смене состояний системы может происходить возврат к предыдущим состояниям, поэтому моделирование пространства состояний будет составлять при упорядочении по хронологии цепочек событий и совпадении момента времени и события (или группы событий) последовательность пар («время», «состояние»). В данном случае (рис. 1) это $e_0 = s_2$, $\{e_1, e_2, e_3, e_4\} =$

$$=\{(t_1, s_5), (t_2, s_4), (t_3, s_1), (t_4, s_3)\}. \quad (3)$$

Событийное функционирование обнаруживается у широкого класса современных систем. В области информационных систем и технологий событийными являются объектно-ориентированные программные системы, сетевые компьютерные, интерактивные, диалоговые системы и другие.

В формальном виде дискретно-событийная система – это некая разновидность временно́го автомата [11], который представляется в следующем виде.

Определение 1. Модель дискретно-событийной системы представляет собой кортеж

$$DS = (X, E, f, \Gamma, x_0), \quad (4)$$

где X – конечное множество, пространство состояний системы; E – конечное множество событий; f – функция смены состояний, $f: X \times E \to X$; Γ – конечное множество активных (и исполняемых в текущий момент) событий; x_0 – начальное состояние.

В связи с тем, что время как таковое не присутствует в модели (4), но при имитационном моделировании все же необходимо воспроизводить хронологию событий по мере упорядоченности их между собой, (4) дополняется модельными часами, связанными с множеством событий. Такие модельные часы представляют собой конечное множество

$$\mathbf{V} = \{\mathbf{v}_i : i \in E\},\ \mathbf{v}_i = \{v_{i,1}, v_{i,2}, \ldots\},$$
$$i \in E,\ v_{i,k} \in \mathbb{R}^+,\ k = 1, 2, \ldots, \quad (5)$$

где $\boldsymbol{v}_{i,k}$ – время жизни (продолжительность) события.

Для последовательности событий $\{e_1, e_2, \ldots, e_k, e_{k+1}, \ldots\}$ можно «включить модельные часы» (5), получить $\mathbf{V} = \{\mathbf{v}_i : i = 1, \ldots, m\}$ и генерировать события $e_{k+1} = h(x_k, \mathbf{v}_1, \ldots, \mathbf{v}_m)$. Динамика состояний дискретно-событийной системы при этом определяется уравнением $x_{k+1} = f(x_k, \mathbf{v}_1, \ldots, \mathbf{v}_m)$.

Таким образом, очень существенной задачей является формирование (генерация) списков событий, то есть пар вида (3), в зависимости от которой можно выделить событийно-ориентированное и процессно-ориентированное исполнение событий. Рассмотрим их более подробно.

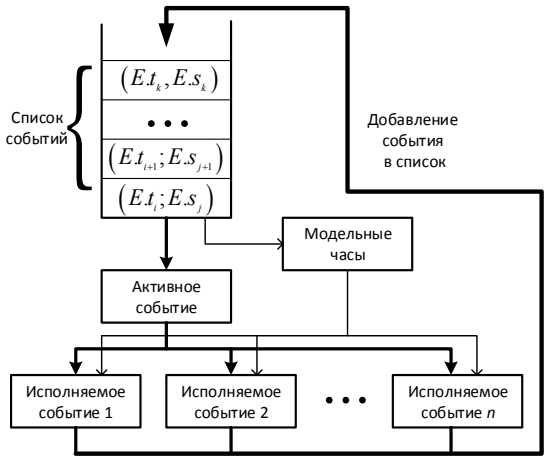

Рис. 2. Событийно-ориентированное моделирование

Fig. 2.Event-based modeling

Событийно-ориентированное моделирование в дискретно-событийной системе проиллюстрировано на рисунке 2.

Список событий является динамической структурой, а модельные часы содержат время последнего исполняемого события. В алгоритмическом виде моделирование заключается в следующей последовательности действий.

Алгоритм 1. Событийно-ориентированное моделирование.

1. Установить модельные часы в 0. Инициализировать начальный список событий, расположив их в хронологическом порядке следования. Элемент списка событий имеет структуру ($E.t_i$; $E.s_j$) и характеризуется временем и типом состояния (t_i, s_j).

2. Выбрать событие E из начала списка. Если список пуст, завершить моделирование.

3. Установить модельные часы в $E.t_i$. Проверить длительность события и при превышении времени, отведенного на моделирование, завершить его.

4. В соответствии с типом события и состояния $E.s_j$ исполнить подпрограмму-обработчик события.

5. Обновить список событий, системные переменные и структуры, поместить новое событие в список событий.

6. Продолжить моделирование, перейдя к п. 2.

Элементы возможной программной реализации алгоритма 1 показаны в примере 1.

Пример 1. Основной событийно-ориентированный обработчик.

```
event_oriented()
 {
    sim_time = 0; // Начальное время = 0
    list_init();  // Инициализация списка событий
    done = FALSE; // Признак завершения моделирования
    while(!done)  // Основной обработчик событий
     {
      next_event(status,time); // Выбор события из списка
   stime = time; // Фиксация времени
```

```
  if (stime > max_sim_time) // Проверка превышения времени
    {
     done = TRUE; // Установка признака завершения
     break; // Выход по завершении
    }
   exec_event(status); // Обработка события
   }
 }
```

Обработчик устанавливает начальное время, инициализирует список событий и в цикле, выбирая следующее событие, переустанавливает модельные часы, проверяя, чтобы не было превышения максимально допустимого времени моделирования, вызывает подпрограмму exec_event(status). При ее реализации имеет смысл запрограммировать хотя бы простейшее планирование списка событий в виде динамической очереди FIFO (First In First Out).

Во втором подходе, процессно-ориентированном исполнении событий, есть возможность исполнения группы событий при их логическом объединении в процессы, как показано на рисунке 3.

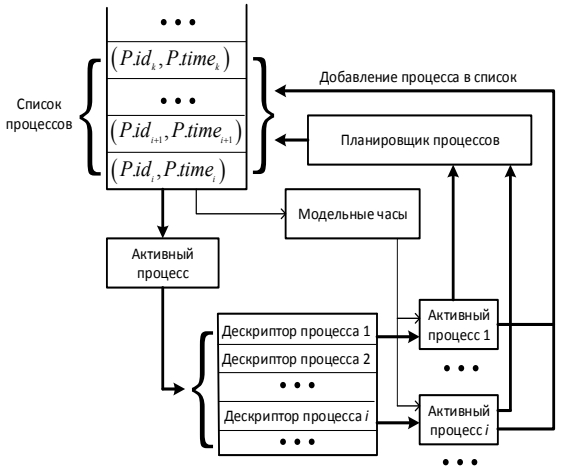

Рис. 3. Процессно-ориентированное моделирование

Fig. 3. Process-based modeling

Основным отличием данного вида от предыдущего является не только возможность объединять процессы в группы, но и планировать переключение между процессами, разделяя при этом совместные ресурсы моделируемой системы. Общая схема такого моделирования представлена в виде алгоритма 2.

Алгоритм 2. Процессно-ориентированное моделирование.

1. Установить модельные часы в 0. Инициализировать начальный список событий, расположив их в хронологическом порядке следования. Элементы списков событий группируются в процессы, имеющие структуру ($P.id_i$; $P.time_i$), и характеризуются идентификатором и временем (*id*; *time*). Запустить основной цикл моделирования.

2. Создать активный процесс *P* из списка планируемых к исполнению событий. Если список пустой, завершить моделирование.

3. Проверить превышение максимального времени, отведенного на моделирование. Если есть превышение, моделирование завершить.

4. Присвоить процессу дескриптор исполнения и передать его планировщику процессов.

5. Выполнить планирование процессов: исполнение, ожидание, переключение, завершение.

6. В рамках процесса требуется исполнить обработчики событий, генерировать новые события.

7. Перейти к п. 2.

В примере 2 приведена схема возможной программной реализации.

Пример 2. Схема процессно-ориентированного обработчика

```
event_job(descriptor) // Планировщик процессов (пример)
 {
    while (TRUE)
    {
     wait(resource); // Ожидать свободные ресурсы
     exec_event(status, time);// Исполнять обработчик события
     switch(descriptor)); // Переключать на другой процесс
     signal(release_event); // Завершить процесс
    }
   }
   process_init() // Инициализация процесса
    {
    for (i=0;i<N;i++)
       event_job(descriptor); // Планировать процесс
    if (stime < max_sim_time)
       {
        next_event(status, time) // Выбор события из списка
        stime = time; Фиксация времени
       }
       else break;
}
   process_oriented()/*Основной процессно-ориентированный обработчик*/
    {
     stime = 0;
     process_init();
     simulate(stime);
}
```

Возможной простейшей стратегией планировщика могут являться FCFS (First Come First Served), «первый пришел – первый обслужен», а также реализация семафорного переключения контекста процессов.

Таким образом, научные исследования в области дискретно-событийного моделирования сейчас достаточно широко развиты в различных направлениях [12–15], а также являются значительным дополнением к методам имитационного моделирования систем. Существуют также программные реализации дискретно-событийных систем моделирования, например AnyLogic, SimPy, SimEvents и другие программные системы.

Дискретно-событийный подход к оценке надежности ПО

В целом дискретно-событийный подход опирается на упомянутые ранее идеи роста надежности ПО, однако в отличие от известных моделей будет использоваться покомпонентная технология моделирования событий возникновения ошибок на входе и выходе программного компонента. Процессы обнаружения и устранения ошибок моделируются случайными точечными процессами, а времена обнаружения ошибок сопоставляются с событиями, при возникновении которых требуется рассчитать вероятностные оценки надежности программных компонентов, которые в зависимости от реализуемых ими алгоритмов будут иметь разные формулы для расчета.

Процедуры моделирования в предлагаемом подходе можно разделить на две группы. Первая группа предназначена для генерации процессов, имитирующих появление ошибок в ПО. Сгенерированные модельные значения разделяются на несколько классов и составляют исходные данные для следующей группы процедур, которая предназначена для оценки системной надежности компонентного ПО. Модельные значения, разделенные на классы, сопоставляются с различными классами ошибок в событийной модели типа «вход–структура–выход». При этом анализируемый с точки зрения надежности программный компонент также относится к одному из классов в зависимости от используемых в нем программных конструкций. Каждому варианту программной конструкции соответствует отдельный вариант расчета оцениваемой надежности ПО.

Рассмотрим подробнее модели первой группы. Определим следующие переменные: $N(t)$ – кумулятивное (накопленное) число ошибок ПО, выявленных на стадиях разработки и тестирования до временного момента t; T_i – i-й интервал времени возникновения либо обнаружения ошибки (T_0=0, i = 1, 2, …); X_i – интервал времени между (i–1)-й и i-й ошибками (X_0 = 0, i = 1, 2, …).

На рисунке 4 показано возникновение события $\{N(t) = i\}$, которое означает, что было обнаружено i ошибок к моменту времени t.

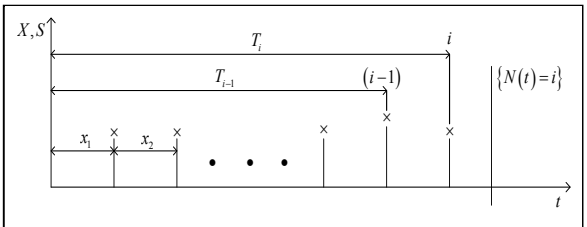

Рис. 4. Дискретно-событийный процесс возникновения и обнаружения ошибок ПО

Fig. 4. A discrete-event process of software bugs occurrence and detection

Таким образом, имеем следующий принцип моделирования: $T_i = \sum_{k=1}^{i} x_k$, $x_i = T_i - T_{i-1}$.

Обозначим $S_i(x)$ уровень ошибок, то есть величину, пропорциональную числу необнаруженных (и, следовательно, неисправленных) ошибок для каждого из интервалов времени x_i, i = 1, 2, …, общий подход к дискретно-событийному моделированию можно определить как

$$S_i(x) = M_i(x)\lambda_i(x), \quad i = 1, 2, ..., N,\ x \ge 0,\ \lambda(x) > 0, \tag{6}$$

где $M_i(x)$ – функция, задающая начальные условия модели; $\lambda(x)$ – функция интенсивности ошибок.

Естественно, в законе моделирования (6) функции $M_i(x)$ и $\lambda_i(x)$ могут быть заданы различными зависимостями – от константы до выражения, имеющего сложную форму. Сама же функция надежности ПО (обозначим ее $R_n(x)$) исходя из (6) также может иметь разный вид, например:

$$R_n(x) = \exp\left[-\int_0^x S_i(x)\,dx\right],\ i = 1, 2, ...,$$

а числовая характеристика, определяющая среднее время между ошибками, может иметь вид

$$E[X_n] = \int_0^\infty R_n(x)dx.$$

Последовательность процедур моделирования, составляющих первую группу методов моделирования, следующая.

1. *Установление исходных положений моделирования.*

1.1. Ошибки ПО возникают в случайные интервалы времени, отсчитываемые последовательно на одной временной оси; имеется некоторое число скрытых ошибок ПО, которые можно обнаружить на этапах разработки и тестирования.

1.2. Ошибка в ПО вызывает ошибочное состояние всей системы и требует ее исправления для восстановления функциональности системы.

1.3. Процесс исправления ошибки выполняется немедленно, новые ошибки при этом не вносятся.

2. *Выбор математического аппарата моделирования.*

Выбираем случайный считающий процесс $N(t)$, $t \ge 0$, подсчитывающий кумулятивное число ошибок в ПО, детектированных ко времени t.

Условия и ограничения следующие: $N(0) = 0$.

2.1. $\{N(t), t \ge 0\}$ – имеет независимые приращения.

2.2. $\Pr\{N(t+\Delta t) - N(t) = 1\} = h(t\Delta t)\mathrm{o}(\Delta t)$, где $h(t)$ – функция мгновенной интенсивности детектирования ошибок.

2.3. $\Pr\{N(t+\Delta t) - N(t) \ge 2\} = \mathrm{o}(\Delta t)$ – в один момент времени обнаруживается не более одной ошибки.

3. *Генерация модельного процесса*.

Генерируем неоднородный случайный пуассоновский процесс:

$$\Pr\{N(t)=n\}=\frac{\{H(t)\}^n}{n!}\exp[-H(t)], \qquad (7)$$

$n = 0, 1, 2, \ldots$, где $H(t)$ – ожидаемое кумулятивное число детектированных ошибок,

$$E[N(t)=n]=H(t)=\int_0^t h(x)dx,$$

где $h(x)$ – интенсивность обнаружения и исправления ошибок.

4. *Переход к численному и алгоритмическому моделированию процессов.*

В упрощенной форме выражение (7) моделируется дискретным неоднородным пуассоновским процессом со средним кумулятивным количеством детектированных ошибок D_n следующим образом:

$$\Pr\{N_x=n\}=\frac{[D_n]^x}{x!}\exp[-D_n], \qquad (8)$$

($x, n = 0, 1, 2, \ldots$).

В дальнейшем вместо выражения (8) могут использоваться другие различные дискретные аналоги выражения (7).

Рассмотрим один из таких аналогов, построенный на основе экспоненциальной модели роста. Пусть H_n обозначает кумулятивное число обнаруженных ошибок за некоторые n периодов эксплуатации ПО, его тестирования, отладки и тому подобное. Процесс роста надежности ПО описывается разностным уравнением

$$H_{n+1}-H_n=\sigma\beta(\alpha-H_n), \qquad (9)$$

где σ – некоторая константа; α – ожидаемое число ошибок на бесконечно длинном интервале времени; β – интенсивность обнаружения ошибок.

Для численной оценки параметров α и β из (9) составим уравнение регрессии: $Y_n = A + BH_n$, где $Y_n = H_{n+1} - H_n$, $A = \delta\alpha\beta$, $B = -\delta\beta$.

Имея статистические данные по числу обнаруженных ошибок ($\hat{A}$ и $\hat{B}$), получаем оценки требуемых параметров из (9): $\hat{\alpha}=-\frac{\hat{A}}{\hat{B}}$ и $\beta=-\frac{\hat{B}}{\delta}$.

Перейдем к процедурам второй группы, где разработана компонентная модель оценки надежности ПО, учитывающая несколько классов ошибок.

5. *Формальное представление компонентной модели оценки надежности ПО.*

Определение 2. Компонентную модель оценки надежности ПО, учитывающую структуру программных конструкций и классы ошибок, определим следующим образом. Имеются три класса ошибок:

1) $F_A=\{F_{A_1},F_{A_2},...,F_{A_K}\}$ – некритические, не воздействующие на выход своего и вход другого блока;

2) $F_B=\{F_{B_1},F_{B_2},...,F_{B_L}\}$ – переходящие, воздействующие на выход своего и вход другого блока;

3) $F_C=\{F_{C_1},F_{C_2},...,F_{C_M}\}$ – критические, воздействующие на выход своего и не воздействующие на вход другого блока.

Оценкой надежности компонента с различной программной конструкцией будем считать вероятность возникновения события, связанного с проявлением ошибки любого вышеуказанного класса, рассматривая программный компонент как систему «вход–структура–выход» $\Pr(I, O)$, $I \in F_B$, $O \in (F_B \cup F_C)$, причем $\sum_{O\in(F_B\cup F_C)}\Pr(I,O)=1, \forall I \in F_B$.

6. *Варианты расчетов возникновения события, сигнализирующего о появлении ошибки ПО.*

В связи с заявленным типом модели «вход–структура–выход» рассмотрим варианты расчета вероятности возникновения события, вызванного ошибкой в зависимости от структуры программного компонента.

6.1. Последовательная структура.

На рисунке 5 показана последовательная структура исполнения алгоритмов A_1, A_2, …, A_n программного компонента.

Вычислительная схема 1.

1. Генерация вероятностей $\Pr_{A_j}(I,F)$, $\Pr_{A_j}(I,O^*)$, $\Pr_{A_j}(I,C)$, $I \in F_B$, $F \in F_A$, $O^* \in F_B$, $C \in F_C$, $j = 1, \ldots, n$.

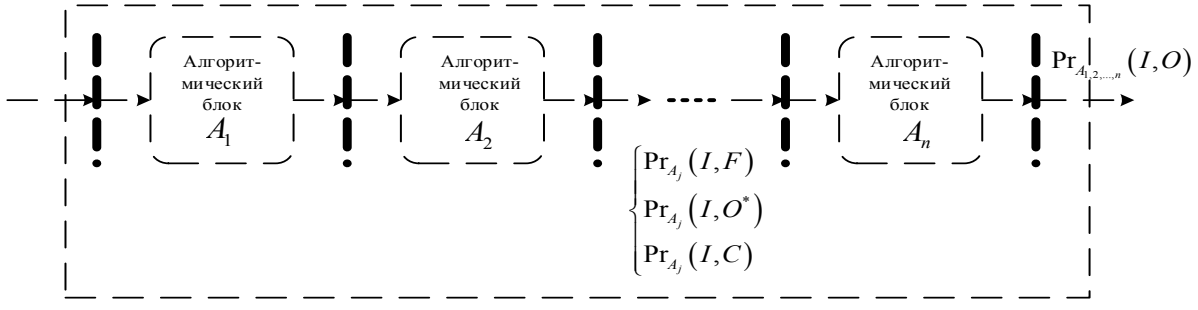

Рис. 5. Последовательная структура программного компонента

Fig. 5. A software component consecutive structure

2. Для имитации возникновения некритической ошибки на выходе A_n: $\Pr_{A_{1,2,\ldots,n}}(I,O)=\Pr_{A_j}(I,F)$, где $F \in F_A$.

3. Для имитации возникновения переходящей ошибки на выходе A_n: $\Pr_{A_{1,2,\ldots,n}}(I,O)= =\sum_{O^*}\Pr_{A_{1,2,\ldots,n-1}}(I,O^*)\Pr_{A_n}(O^*,O)$, где $O^* \in F_B$.

4. Для имитации возникновения критической ошибки на выходе A_n: $\Pr_{A_{1,2,\ldots,n}}(I,O)=\Pr_{A_{1,2,\ldots,n-1}}(I,C)+ +\sum_{O^*\in F_B}\Pr_{A_{1,2,\ldots,n-1}}(I,O^*)\Pr_{A_n}(O^*,O)$, где $O^*\in F_B$, $C\in F_C$.

6.2. Разветвляющаяся структура.

Если программный компонент имеет структуру разветвления с c_n ветвями, из которых первые $n-1$ являются ветвями «если–то», а n-я ветвь общим «иначе», то расчет будет производиться по вычислительной схеме 2 (рис. 6).

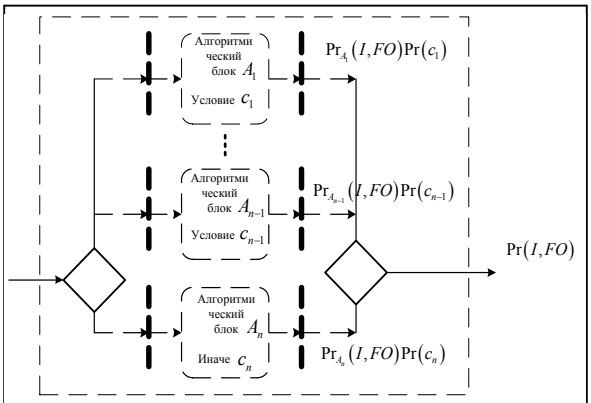

Рис. 6. Разветвляющаяся структура программного компонента

Fig. 6. A software component branching structure

Вычислительная схема 2.

1. Генерация вероятностей $\Pr_{A_j}(I,FO)$, $I \in F_B$, $FO \in (F_B \cup F_C)$, $j = 1, \ldots, n$, $\Pr(c_i)$, $i = 1, \ldots, n-1$.

2. Для имитации возникновения ошибки рассчитывается $\Pr(I,O)=\sum_{i=1}^{n-1}\Pr_{A_i}(I,FO)\Pr(c_i)+ +\Pr_{A_n}(I,FO)\left(1-\sum_{i=1}^{n-1}\Pr(c_i)\right)$.

6.3. Циклическая структура.

Если программный компонент имеет циклическую структуру, ее можно трансформировать в n-кратное повторение последовательной структуры исполнения алгоритма $\underbrace{A_1, A_1, \ldots, A_1}_{n \text{ раз}}$ программного компонента. Таким образом, для циклической структуры оказывается подходящей вычислительная схема из п. 6.1.

6.4. Параллельная структура.

Если программный компонент предусматривает параллельное исполнение n алгоритмов A_1, A_2, …, A_n, то моделируется возникновение событий,

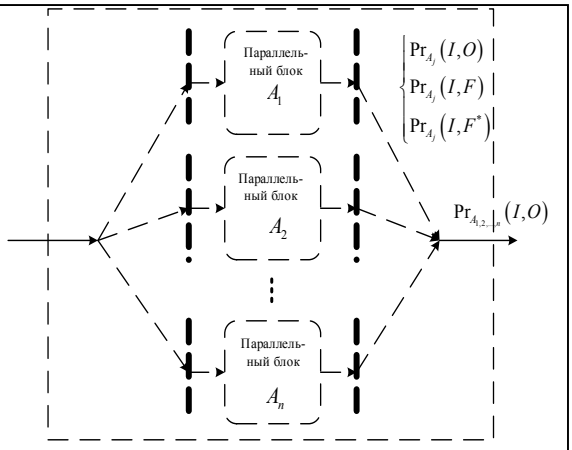

Рис. 7. Параллельная структура программного компонента

Fig. 7. A software component parallel structure

связанных с классами некритичных и критичных ошибок. Структура исполнения алгоритмов в этом случае показана на рисунке 7.

Вычислительная схема 3.

1. Генерация вероятностей $\Pr_{A_j}(I,O)$, $\Pr_{A_j}(I,F)$, $\Pr_{A_j}(I,F^*)$, $I \in F_B$, $F^* \in F_A$, $F \in (F_A \cup F_C)$ $j = 1, \ldots, n$.

2. Если $n-1$ параллельных ветвей имеют корректный выход, а n-я ветвь имеет некритическую ошибку, то рассчитывается $\Pr_{A_{1,2,\ldots,n}}(I,O)= =\Pr_{A_{1,2,\ldots,n-1}}(I,O)\Pr_{A_n}(I,F^*)$.

3. Если $n-1$ параллельных ветвей имеют некритическую ошибку, а n-я ветвь имеет критическую ошибку либо наоборот, то рассчитывается $\Pr_{A_{1,2,\ldots,n}}(I,O)=\left[\sum_{F\in(F_A\cup F_C)}\Pr_{A_{1,2,\ldots,n-1}}(I,F)\right]\Pr_{A_n}(I,F)$.

На основании изложенного сделаем следующие выводы. В статье предложен дискретно-событийный подход к оценке надежности ПО, использующий покомпонентную технологию моделирования событий возникновения ошибок на входе и выходе программного компонента. Процессы обнаружения и устранения ошибок моделируются случайными точечными процессами, а времена обнаружения ошибок сопоставляются с событиями, при возникновении которых требуется рассчитать вероятностные оценки надежности программных компонентов, которые в зависимости от реализуемых ими алгоритмических структур имеют разные формулы для расчета.

Литература

1. Banks J. Discrete-event system simulation. Pearson Prentice Hall, 2005, 608 p.

2. Tyszer J. Object-oriented computer simulation of discrete-event systems. Springer US, 1999, 258 p.

3. Wainer G.A. Discrete event modeling and simulation: a practitioner's approach. CRC Press, 2009, 486 p.

4. Лоу А.М., Кельтон В.Д. Имитационное моделирование.

Классика CS. 3-е изд. СПб: Питер; Киев: BHV, 2004. 847 с.

5. Рыжиков Ю.И. Имитационное моделирование. Теория и технологии. М.: Альтекс-А, 2004. 384 с.

6. Финаев В.И. Алгоритмизация и имитационное моделирование с применением аппарата систем массового обслуживания: учеб. пособие. Таганрог: Изд-во ТРТУ, 2003. 155 с.

7. Таха Х.А. Введение в исследование операций. 7-е изд.; [пер. с англ.]. М.: Вильямс, 2005. 912 с.

8. Бражник А.Н. Имитационное моделирование: возможности GPSS WORLD. СПб: Реноме, 2006. 439 с.

9. Котов В.Е. Сети Петри. М.: Наука, 1984. 160 с.

10. Hruz B., Zhou M.C. Modeling and Control of Discrete-event Dynamic Systems: with Petri Nets and Other Tools. Springer London, 2007, 341 p.

11. Cassandras C.G., Lafortune S. Introduction to discrete event systems. 2nd Edition. Springer, 2008, 782 p.

12. Baccelli F., Cohen G., Olsder G.J., Quadrat J.-P. Synchronization and linearity: an algebra for discrete event systems. Chichester: Wiley, 1992, 514 p.

13. Gondran M., Minoux M. Graphs, Dioids and Semirings. New Models and Algorithms. Springer Science+Business Media, LLC. 2008, 384 p.

14. Heidergott B., Jan Olsder G., van der Woude J. Max Plus at work. Modeling and analysis of synchronized systems: A course on Max-Plus Algebra and Its Applications. Princeton Univ. Press, Princeton. 2006, 232 p.

15. Wainer G.A. Discrete Event Modeling and Simulation: A practitioner's approach. CRC Press, 2009, 486 p.

DOI: 10.15827/0236-235X.112.158-165 Received 09.09.15

SOFTWARE RELIABILITY EVALUATION BY DISCRETE-EVENT SIMULATION METHODS

(The work has been done with financial support from RFBR within the research projects no. 15-08-01886-a, 15-01-03067-a)

***Butakova M.A.**, Dr.Sc. (Engineering), Professor, Dean, butakova@rgups.ru;*
***Guda A.N.**, Dr.Sc. (Engineering), Professor, Vice-rector, guda@rgups.ru;*
***Chernov A.V.**, Head of Chair, Dr.Sc. (Engineering), Professor, avche@yandex.ru;*
***Chubeyko S.V.**, Ph.D. (Engineering), greyc@mail.ru*
(Rostov State Transport University,
PL Rostovskogo Strelkovogo Polka Narodnogo Opolcheniya 2, Rostov-on-Don, 344038, Russian Federation)

Abstract. The article considers discrete-event modeling and presents its distinctive features comparing to other types of modeling. The main difference is a lack of time reference, i.e. it is enough to keep the sequence of events, and thus the time interval between events isn't important. The authors give the definition of a discrete-event system model and add model time, which reproduce the course of events. The article solves the important problem of the event list generation in various ways: object-based and the process-based execution of events. Both ways are considered in detail with the illustration, algorithm and a program implementation element. Events can be united into groups called processes. The process-based modeling is more difficult than object-based as there is a process scheduler. The article also considers the assessment of software reliability based on discrete-event approach. This approach is based on the idea of software reliability growth. Bug scanning is modeled by a stochastic point process. A detected bug is eliminated, thereby the software becomes more reliable. Modeling has two parts: generation of the processes, which imitate introducing bugs in the software, and an assessment of component software system reliability. The paper considers options of calculating bug probability depending on program structure: consecutive, branching, cyclic and parallel. Each option has an illustration and a computation scheme. The cyclic scheme of a software component has the computation scheme of a consecutive component as it is some kind of same-type repetitions of a software component consecutive structure.

Keywords: discrete-event simulation, event-based modeling, process-based modeling, software reliability, software structure.

References

1. Banks J. *Discrete-event System Simulation*. Pearson Prentice Hall Publ., 2005, 608 p.

2. Tyszer J. *Object-oriented Computer Simulation of Discrete-event Systems*. Springer US Publ., 1999, 258 p.

3. Wainer G.A. *Discrete Event Modeling and Simulation: A Practitioner's Approach.* CRC Press, 2009, 486 p.

4. Lou A., Kelton V. *Simulation. Classics CS*. 3th ed., St.-Petersburg, Piter, Kiev, BHV Publ., 2004, 847 p.

5. Ryzhikov U. *Imitatsionnoe modelirovanie. Teoriya i tekhnologii* [Simulation. Theory and Technology]. Moscow, Altex-A Publ., 2004, 384 p.

6. Finaev V.I. *Algoritmizatsiya i imitatsionnoe modelirovanie s primeneniem apparata sistem massovogo obsluzhivaniya* [Algorithmic and Simulation using Queueing Theory]. Study guide, Taganrog, Taganrog State Univ. of Radio Engineering Publ., 2003, 155 p.

7. Taha H.A. *Operations Research: an Introduction*. 7th ed. 2002, Prentice Hall Publ., 848 p. (Russ. ed.: Vilyams Publ., 2005, 912 p.).

8. Brazhnik A. *Imitatsionnoe modelirovanie: vozmozhnosti GPSS WORLD* [Simulation: GPSS WORLD capabilities]. St.-Petersburg, Renome Publ., 2006, 439 p.

9. Kotov V. *Seti Petri* [Petri Nets]. Moscow, Nauka Publ., 1984, 160 p.

10. Hruz B., Zhou M.C. *Modeling and Control of Discrete-event Dynamic Systems: with Petri Nets and Other Tools.* Springer London Publ., 2007, 341 p.

11. Cassandras C.G., Lafortune S. *Introduction to Discrete Event Systems*. 2nd Edition. Springer Publ., 2008, 782 p.

12. Baccelli F., Cohen G., Olsder G.J., Quadrat J.-P. *Synchronization and linearity: An algebra for discrete event systems*. Chichester, Wiley Publ., 1992, 514 p.

13. Gondran M., Minoux M. Graphs, Dioids and Semirings. *New Models and Algorithms*. Springer Science+Business Media Publ., 2008, 384 p.

14. Heidergott B., Jan Olsder G., van der Woude J. *Max Plus at work. Modeling and analysis of synchronized systems*: A course on Max-Plus Algebra and Its Applications. Princeton Univ. Press, Princeton, 2006, 232 p.

15. Wainer G.A. *Discrete Event Modeling and Simulation*: A Practitioner's Approach. CRC Press, 2009, 486 p.