Оценка надежности программного обеспечения методами дискретно-событийного моделирования

Article · December 2015

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

CITATIONS
0

READS
6

8 authors, including:

Maria Butakova
Rostov State Transport University
9 PUBLICATIONS **4** CITATIONS
SEE PROFILE

Andrey Chernov
Rostov State Transport University, Rostov-on…
18 PUBLICATIONS **36** CITATIONS
SEE PROFILE

УДК 658.012.011:656.2.001.57+06

ОЦЕНКА НАДЕЖНОСТИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ МЕТОДАМИ ДИСКРЕТНО-СОБЫТИЙНОГО МОДЕЛИРОВАНИЯ

Бутакова М.А., д.т.н., профессор; Гуда А.Н., д.т.н., профессор;
Чернов А.В., д.т.н., профессор; Чубейко С.В., к.т.н.

SOFTWARE RELIABILITY EVALUATION BY DISCRETE-EVENT SIMULATION METHODS

Butakova M.A., Guda A.N., Chernov A.V., Chubeyko S. V.

Аннотация

В статье рассматривается дискретно-событийное моделирование и представлены его отличительные особенности от других видов моделирования. Основное отличие – это 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. В статье также решается важная задача генерации списка событий различными способами: объектно-ориентированное и процессно-ориентированное исполнение событий. Далее подробно рассматриваются оба способа: приводится 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 программного обеспечения, базирующаяся на дискретно-событийном подходе. Данный подход основан на идее роста надежности программного обеспечения. Поиск ошибок моделируется случайным точечным процессом. При обнаружении ошибки она устраняется, тем самым программное обеспечение становится более надежным. Моделирование делится на две части: генерация процессов, имитирующих появления ошибок в программном обеспечении, и оценка системной надежности компонентного программного обеспечения. 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.

Annotation

In article discrete and event modeling is considered and its distinctive features from other types of modeling are presented. The main difference is a lack of a binding at the right time, it is enough to observe sequence of approach of events, thus it isn't important, what time interval will be between events. Definition of model of discrete and event system with addition with its model time which reproduce chronology of events is given. In article the important problem of generation of the list of events is also solved in various ways: object-oriented and the process oriented execution of events. Further in detail both ways are considered: the illustration, algorithm and an element of program realization is given. Events can unite in groups which are called as processes. The process oriented modeling is more difficult than object-oriented as the scheduler of processes is required. Also in article the assessment of reliability of the software which is based on discrete and event approach is considered. This approach is based on idea of growth of reliability of the software. Search of software bugs is modelled by stochastic point process. At detection of a bug it is eliminated, thereby the software becomes more reliable. Modeling shares on two parts: generation of the processes imitating emergence of bugs in the software, and an assessment of system reliability of the component software. In article options of calculation of probability of emergence of a bug depending on structure of programs are considered: the consecutive, branching, cyclic and parallel structure of a program component. For each option the illustrating drawing is presented and the computing scheme is provided. For the cyclic scheme of a program component the computing scheme of a consecutive component as it is some kind of same repetitions of consecutive structure of a program component is used.

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

Введение

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, 2, 3] существенно отличают рассматриваемый далее 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 являются нелинейными 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) [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 являются (часто усредненные) времена пребывания заявки в очереди, системе, времена обслуживания, вероятности пребывания в очереди, вероятности обслуживания (за некоторый период) и другие вероятностные и статистические характеристики.

Заметим следующее, 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 нами далее подходу с идейной стороны, но не со стороны реализации методов моделирования. В целом, процессное время означает, что изменение состояний системы, а также её модели можно отметить на некоторой временной шкале, если шкала является непрерывной, то естественным будет непрерывное моделирование состояний системы, иначе – дискретное моделирование состояний системы, 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 их точной привязки к временной шкале. Существенными являются лишь факты наличия 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: $\mathbf{u}(t)$– входа, $\mathbf{x}(t)$ – состояния, $\mathbf{y}(t)$ – выхода, динамика моделируемой системы описывается уравнениями:

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

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

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

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

$$\text{.....} \quad ,$$

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

и *k* выходных уравнений системы

$$y_1(t) = g_1\left(x_1(t),...,x_n(t),u_1(t),...,u_m(t),t\right)$$
$$\text{.....} \quad ,$$
$$y_k(t) = g_k\left(x_1(t),...,x_n(t),u_1(t),...,u_m(t),t\right)$$

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

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 его измерениями мы бы 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 рис.1. В верхней части показано пространство состояний непрерывной системы, в средней части – дискретной системы, а в нижней части – дискретно-событийной системы. Ясно, что пространство состояний дискретно-событийной системы является дискретным и составляет события $s_1,...,s_5$, а переключение между этими состояниями происходит в соответствии наступлением некоторых событий $e_1,...,e_4$. Естественным образом, 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 для рис. 1. это

$$e_0 = s_2,\ \{e_1,e_2,e_3,e_4\} = \{(t_1,s_5),(t_2,s_4),(t_3,s_1),(t_4,s_3)\}. \qquad (3)$$

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

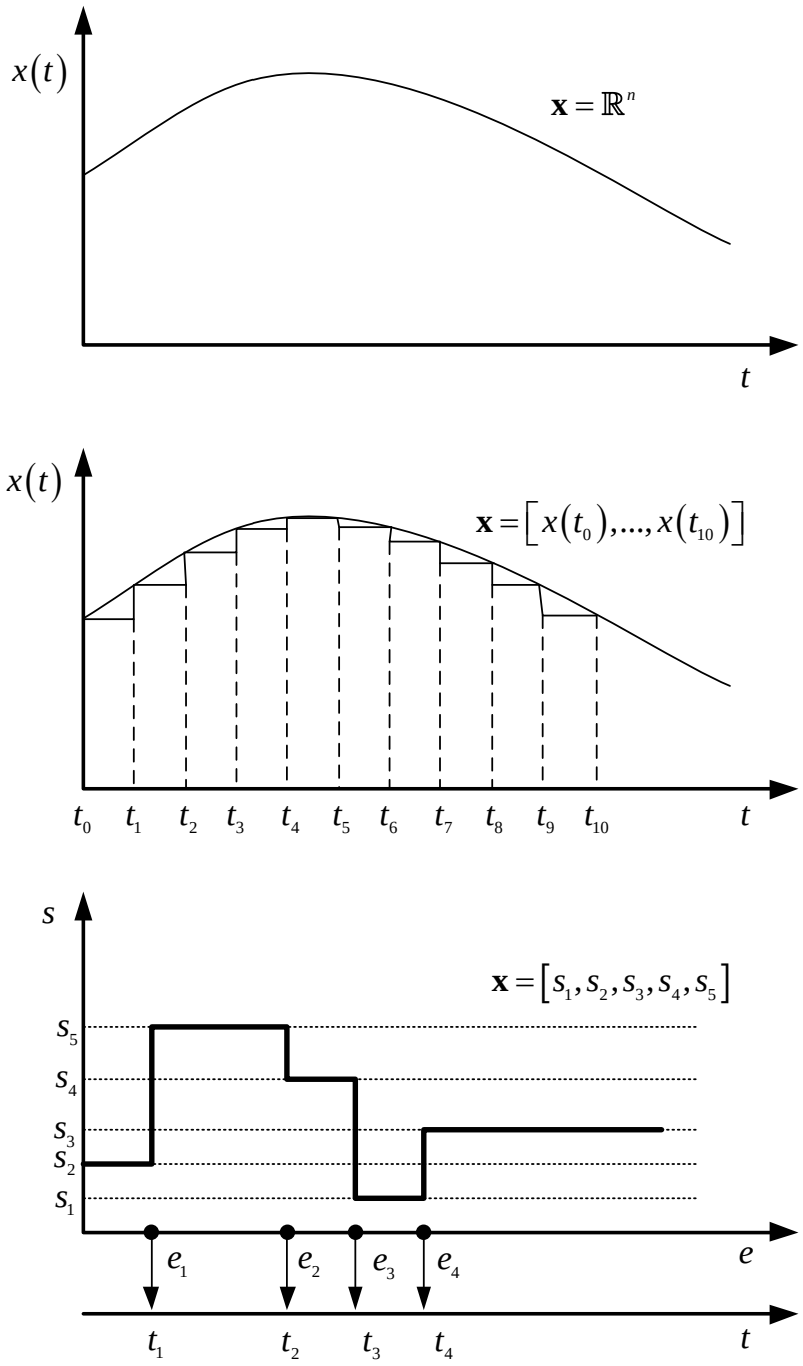

Рис. 1 – Непрерывная, дискретная и дискретно-событийная система

В формальном виде дискретно-событийная система представляет собой некоторую разновидность временно́го автомата [11], который представляется в следующем виде.

О п р е д е л е н и е 1 .

Модель дискретно-событийной системы представляет собой кортеж

$$DS = (X, E, f, \Gamma, x_0), \tag{4}$$

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, то (4) дополняется «модельными часами», связанными с множеством событий. Такие «модельные часы» представляют собой конечное множество

$$\mathbf{V} = \{\mathbf{v}_i : i \in E\},\ \mathbf{v}_i = \{v_{i,1}, v_{i,2}, ...\},\ i \in E,\ v_{i,k} \in \mathbb{R}^+,\ k = 1, 2, ...\ , \tag{5}$$

где $v_{i,k}$ – время жизни (продолжительность) события.

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

$x_{k+1} = f(x_k, \mathbf{v}_1, ..., \mathbf{v}_m)$.

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

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

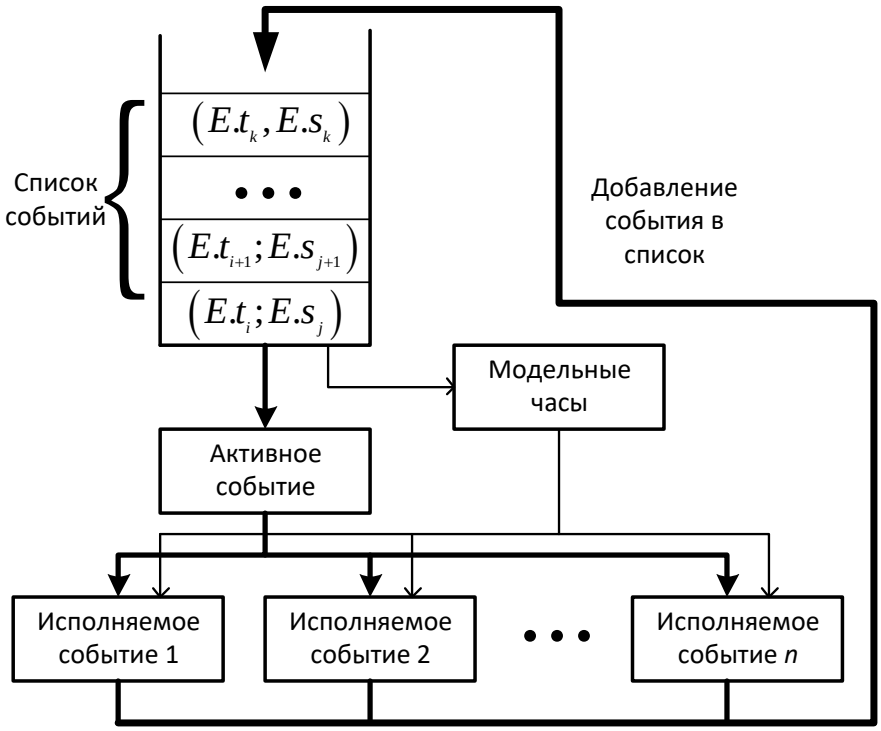

Рис. 2 – Событийно-ориентированное моделирование

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. Выбирать событие E из начала списка. Если список пуст, то завершить моделирование.
3. Установить модельные часы в $E.t_i$. Проверить длительность события и при превышении времени, отведенного на моделирование его завершить.
4. В соответствии с типом события и состояния $E.s_j$ исполнить подпрограмму-обработчик события.
5. Обновлять список событий, системные переменные и структуры, помещать новое событие в список событий.
6. Продолжать моделирование, переходя 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 завершению
    }
   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 её 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 рис. 3.

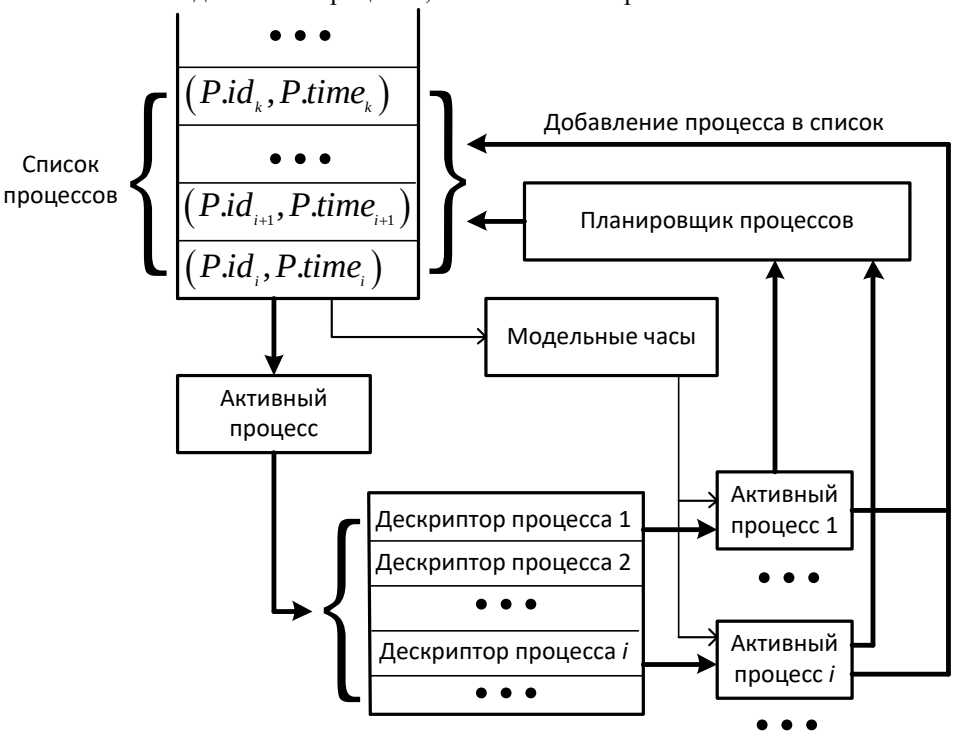

Рис. 3 – Процессно-ориентированное моделирование

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 группируется в процессы, имеющие структуру $(P.id_i; P.time_i)$ и характеризуется идентификатором и временем $(id; time)$. Запустить основной цикл моделирования.
2. Создать активный процесс P из списка планируемых к исполнению событий. Если список пустой, то завершить моделирование.
3. Проверять превышение максимального времени, отведенного на моделирование. Если есть превышение, то моделирование завершить.
4. Присвоить процессу дескриптор исполнения и передать его планировщику процессов.
5. Выполнять планирование процессов: исполнение, ожидание, переключение, завершение.
6. В рамках процесса исполнять 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 может являться *FCFS* (*First Come First Served*), «первый пришел – первый обслужен», а также реализация семафорного переключения контекста процессов.

В итоге этого раздела статьи следует заметить, что научные исследования в области дискретно-событийного моделирования сейчас достаточно широко развиты в различных направлениях [12, 13, 14, 15], а также служат как существенное дополнение методов 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 программного обеспечения

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, которые разделены 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:

1) $N(t)$ – кумулятивное (накопленное) число ошибок ПО, выявленное на стадиях разработки и тестирования до временного момента t;

2) T_i – i-й интервал времени возникновения, либо обнаружения ошибки $(T_0 = 0,\ i = 1,2,...)$;

3) X_i – интервал времени между $(i-1)$ и i-й ошибками $(X_0 = 0,\ i = 1,2,...)$.

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

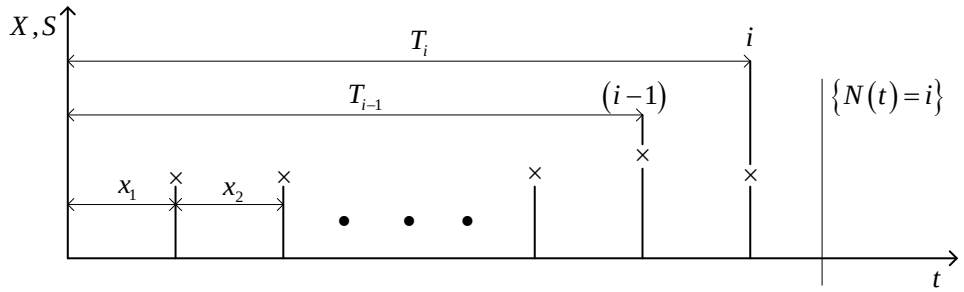

Рис. 4 – Дискретно-событийный процесс возникновения и обнаружения ошибок ПО

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, т.е. величину, пропорциональную числу не обнаруженных (и, следовательно, не исправленных) ошибок для каждого из интервалов времени x_i, $i = 1,2,...$, общий подход к дискретно-событийному моделированию можно определить как

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

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, начиная от константы и 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, новых ошибок при этом не вносится.

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.

2.3 $Pr\{N(t+\Delta t) - N(t) = 1\} = h(t\Delta t)\,o(\Delta t)$ – где $h(t)$ – функция мгновенной интенсивности детектирования ошибок.

2.4 $Pr\{N(t+\Delta t) - N(t) \ge 2\} = o(\Delta t)$ – в один момент времени обнаруживается не более одной ошибки.

3. *Генерация модельного процесса*.

Генерируем неоднородный случайный пуассоновский процесс

$$Pr\{N(t) = n\} = \frac{\{H(t)\}^n}{n!}\exp[-H(t)],\ n = 0,1,2,... \qquad (7)$$

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:

$$Pr\{N_x = n\} = \frac{[D_n]^x}{x!}\exp[-D_n],\ (x, n = 0,1,2,...). \quad (8)$$

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:

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

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 наблюдаемые статистические данные по числу обнаруженных ошибок (назовем их $\hat{A}$ и $\hat{B}$) можно получить оценки требуемых параметров из (9): $\hat{\alpha} = -\frac{\hat{A}}{\hat{B}}$ и $\beta = -\frac{\hat{B}}{\delta}$.

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

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

О п р е д е л е н и е 2 .

Компонентную модель оценки надежности ПО, учитывающую структуру программных конструкций и классы ошибок определим следующим образом. Имеется 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 блока 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 появление 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 рис. 5 показана последовательная структура исполнения алгоритмов $A_1, A_2, ..., A_n$ программного компонента.

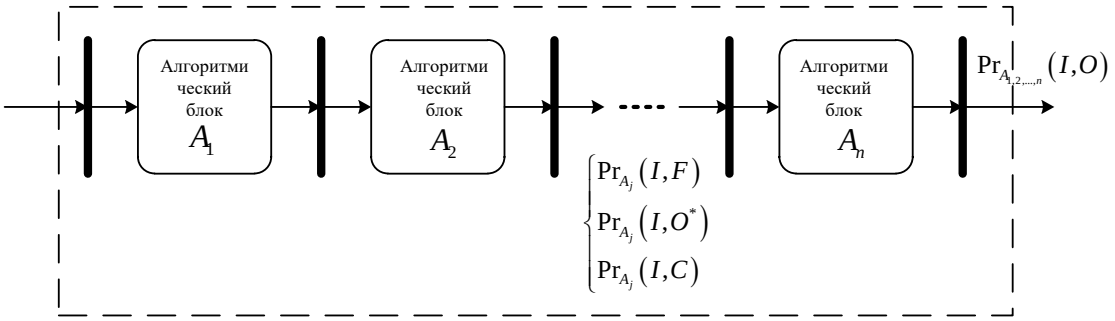

Рис. 5 – Последовательная структура программного компонента

Вычислительная схема 1:

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

2. Для имитации возникновения некритической ошибки на выходе A_n:

$\Pr_{A_{1,2,...,n}}(I,O) = \Pr_{A_j}(I,F)$, где $F \in F_A$.

3. Для имитации возникновения переходящей ошибки на выходе A_n:

$\Pr_{A_{1,2,...,n}}(I,O) = \sum_{O^*} \Pr_{A_{1,2,...,n-1}}(I,O^*)\Pr_{A_n}(O^*,O)$, где $O^* \in F_B$.

4. Для имитации возникновения критической ошибки на выходе A_n:

$\Pr_{A_{1,2,...,n}}(I,O) = \Pr_{A_{1,2,...,n-1}}(I,C) + \sum_{O^* \in F_B} \Pr_{A_{1,2,...,n-1}}(I,O^*)\Pr_{A_n}(O^*,O)$,

где $O^* \in F_B, C \in F_C$.

6.2. *Разветвляющаяся структура*

Если программный компонент имеет структуру разветвления, имеющую c_n ветвей, из которых первые $n-1$ являются ветвями «если-то», а n -я ветвь является общим «иначе», то расчет будет производиться по нижеследующей вычислительной схеме. Иллюстрация показана на рис. 6.

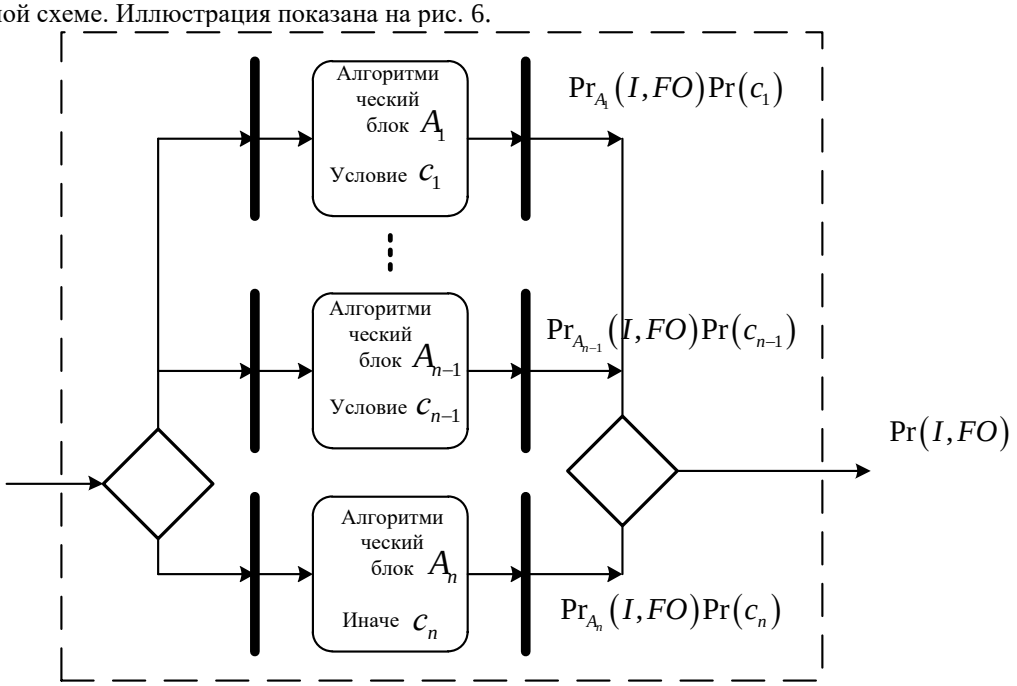

Рис.6 Разветвляющаяся структура программного компонента

Вычислительная схема 2:

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

$\Pr(c_i)$, $i = 1,...,n-1$.

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, то её можно трансформировать в n -кратное повторение последовательной структуры исполнения алгоритма $\underbrace{A_1, A_1, ..., A_1}_{n\ раз}$ 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, связанных с классами некритичных и критичных ошибок. Структура исполнения алгоритмов в этом случае представлена на рис. 7.

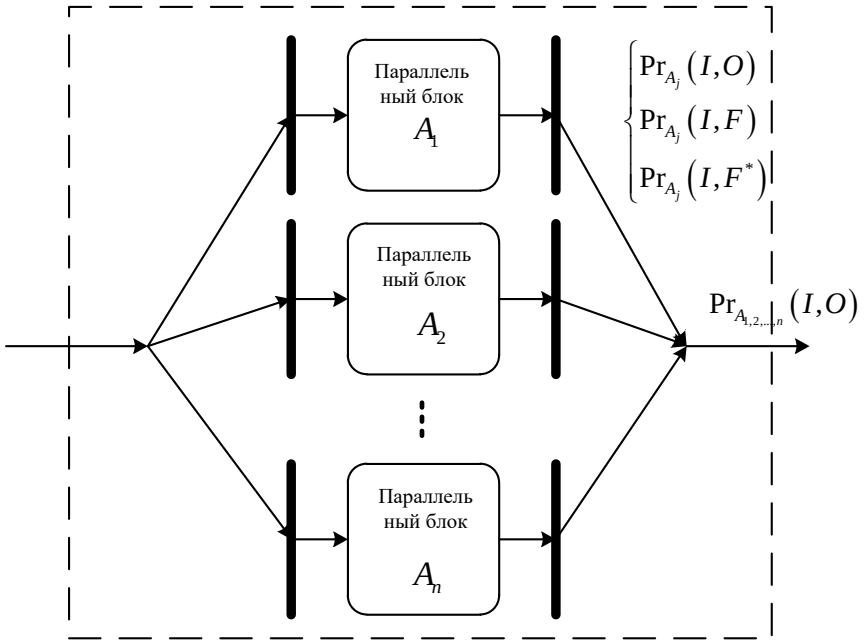

Рис. 7. Параллельная структура программного компонента

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

$$\Pr_{A_{1,2,\ldots,n}}(I,O) = \left[\sum_{F\in(F_A\cup F_C)} \Pr_{A_{1,2,\ldots,n-1}}(I,F)\right]\Pr_{A_n}(I,F).$$

Выводы

В статье предложен дискретно-событийный подход к оценке надежности программного обеспечения 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. Работа выполнена при финансовой поддержке РФФИ, проекты 15-08-01886-а, 15-01-03067-а.

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: Издательская группа 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. Учебное пособие. Изд-во ТРТУ, Таганрог, 2003. – 155 с.
7. Таха Х.А. Введение в исследование операций. 7-е издание: Пер. с англ. – Издательский дом «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. Second Edition. – Springer, 2008. – 782 p.
12. Baccelli F. Synchronization and linearity: An algebra for discrete event systems / Baccelli F., Cohen G., Olsder G.J., Quadrat J.-P. – Chichester: Wiley. – 1992. – 514 p.

13. Gondran M. Graphs, Dioids and Semirings. New Models and Algorithms. / Gondran M., Minoux M. – Springer Science+Business Media, LLC. – 2008. 384 p.
14. Heidergott B. Max Plus at Work. Modeling and Analysis of Synchronized Systems: A course on Max-Plus Algebra and Its Applications // Heidergott B., Jan Olsder G., van der Woude J. – Princeton University Press, Princeton. – 2006. – 232 p.
15. Wainer G.A. Discrete Event Modeling and Simulation: A Practitioner's Approach. – CRC Press, 2009. – 486 p.

Literature

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. Lou A., Kelton V. Simulation. Classics CS 3th ed. SPb.: Piter; Kiev: Publishing Group BHV, 2004. – 847 p.
5. Rizhikov U. Simulation. Theory and Technology M.: Altex-A, 2004. – 384 p.
6. Финаев В.И. Finaev V. Algorithmic and Simulation using Queueing Theory. Study guide. Ed. TRTU, Taganrog, 2003. – 155 p.
7. Taha H. Introduction to Operations Research. 7th ed: The lane with Englis. –Publishing house «Vilyms», 2005. – 912 p.
8. Brazhnik A. Simulatio: opportunities GPSS WORLD. — SPb..: Renome, 2006. – 439 p.
9. Kotov V. Petri nets.- M.: science, 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, 2007. – 341 p/
11. Cassandras C.G., Lafortune S. Introduction to Discrete Event Systems. Second Edition. – Springer, 2008. – 782 p.
12. Baccelli F. Synchronization and linearity: An algebra for discrete event systems / Baccelli F., Cohen G., Olsder G.J., Quadrat J.-P. – Chichester: Wiley. – 1992. – 514 p.
13. Gondran M. Graphs, Dioids and Semirings. New Models and Algorithms. / Gondran M., Minoux M. – Springer Science+Business Media, LLC. – 2008. 384 p.
14. Heidergott B. Max Plus at Work. Modeling and Analysis of Synchronized Systems: A course on Max-Plus Algebra and Its Applications // Heidergott B., Jan Olsder G., van der Woude J. – Princeton University Press, Princeton. – 2006. – 232 p.
15. Wainer G.A. Discrete Event Modeling and Simulation: A Practitioner's Approach. – CRC Press, 2009. – 486 p.

Сведения об авторах

Бутакова Мария Александровна, доктор технических наук, профессор кафедры «Информатика», декан факультета «Информационные технологии управления» Ростовского государственного университета путей сообщения.
butakova@rgups.ru
Контактный телефон: 8-918-597-66-71
Адрес: г. 344038, Ростов-на-Дону, пл. им. Ростовского Стрелкового полка Народного ополчения, 2, Ростовский государственный университет путей сообщения

Гуда Александр Николаевич, доктор технических наук, профессор, проректор по научной работе Ростовского государственного университета путей сообщения.
guda@rgups.ru
Контактный телефон: (863)2-726-350
Адрес: г. 344038, Ростов-на-Дону, пл. им. Ростовского Стрелкового полка Народного ополчения, 2, Ростовский государственный университет путей сообщения

Чернов Андрей Владимирович, доктор технических наук, профессор, заведующий кафедрой «Вычислительная техника и автоматизированные системы управления» Ростовского государственного университета путей сообщения.
avche@yandex.ru

Контактный телефон: (863)2-726-242
Адрес: г. 344038, Ростов-на-Дону, пл. им. Ростовского Стрелкового полка Народного ополчения, 2, Ростовский государственный университет путей сообщения

Чубейко Сергей Валерьевич, ассистент кафедры «Информатика» Ростовского государственного университета путей сообщения.
greyc@mail.ru
Контактный телефон: 8-951-524-97-00
Адрес: г. 344038, Ростов-на-Дону, пл. им. Ростовского Стрелкового полка Народного ополчения, 2, Ростовский государственный университет путей сообщения, кафедра «Информатики»

Authors

Butakova, Mariya A., Dr Tech. Science, professor, professor of the Informatics Department, dean of the faculty of Information technology of management, Rostov State Transport University
butakova@rgups.ru
Address: 344038, Russia, Rostov-na-Donu, sq. Rostovskogo Strelkovogo polka Narodnogo Opolchenija, 2

Guda, Alexander N. Dr Tech. Science, professor, head of the Informatics Department, vice-rector, Rostov State Transport University
guda@rgups.ru
Address: 344038, Russia, Rostov-na-Donu, sq. Rostovskogo Strelkovogo polka Narodnogo Opolchenija, 2

Chernov, Andrey V. Dr Tech. Science, professor, head of the Computers and Automated Control Systems Department, Rostov State Transport University
avche@yandex.ru
Address: 344038, Russia, Rostov-na-Donu, sq. Rostovskogo Strelkovogo polka Narodnogo Opolchenija, 2

Chubeyko, Sergey V., PhD, assistant of the Informatics Department, Rostov State Transport University
greyc@mail.ru

Address: 344038, Russia, Rostov-na-Donu, sq. Rostovskogo Strelkovogo polka Narodnogo Opolchenija, 2