ТЕОРЕТИЧЕСКИЙ И ПРИКЛАДНОЙ НАУЧНО-ТЕХНИЧЕСКИЙ ЖУРНАЛ

Издается с сентября 2010 г.

Журнал издается при поддержке Отделения математических наук РАН, Отделения нанотехнологий и информационных технологий РАН, МГУ имени М.В. Ломоносова, МГТУ имени Н.Э. Баумана, ОАО "Концерн “Сириус"

СОДЕРЖАНИЕ

Журнал зарегистрирован в Федеральной службе по надзору в сфере связи, информационных технологий и массовых коммуникаций. Свидетельство о регистрации ПИ № ФС77-38590 от 24 декабря 2009 г.

Журнал распространяется по подписке, которую можно оформить в любом почтовом отделении (индексы: по каталогу агентства "Роспечать" — **22765**, по Объединенному каталогу "Пресса России" — **39795**) или непосредственно в редакции.
Тел.: (499) 269-53-97. Факс: (499) 269-55-10.
Http://novtex.ru/pi.html E-mail: prin@novtex.ru
Журнал включен в систему Российского индекса научного цитирования.
Журнал входит в Перечень научных журналов, в которых по рекомендации ВАК РФ должны быть опубликованы научные результаты диссертаций на соискание ученой степени доктора и кандидата наук.

THEORETICAL AND APPLIED SCIENTIFIC AND TECHNICAL JOURNAL

SOFTWARE ENGINEERING

PROGRAMMNAYA INGENERIA

№ 7
July
2015

Published since September 2010

CONTENTS

Information about the jornal is available online at:
http://novtex.ru/pi.html, e-mail:prin@novtex.ru

УДК 004.2 + 004.4

С. Г. Елизаров, канд. физ.-мат. наук, ст. науч. сотр., МГУ им. М.В. Ломоносова,
Г. А. Лукьянченко, аспирант, НИЦ "Курчатовский институт", г. Москва
В. В. Корнеев, д-р техн. наук, проф., зам. директора по научной работе,
e-mail: korv@rdi-kvant.ru, ФГУП "НИИ "Квант", г. Москва

Технология параллельного программирования экзафлопсных компьютеров

Предложена модель вычислений и организации памяти для экзамасштабных суперкомпьютеров. Описаны основные конструкции языка программирования и аппаратные блоки, необходимые для поддержки предложенной модели. Представлен действующий прототип, разработанный на программируемой логической интегральной схеме узла суперкомпьютера экзафлопсного уровня, содержащий более тысячи вычислительных ядер. Прототип допускает гибкую настройку вычислительной архитектуры под задачи пользователя и предлагает для решения параллельных задач такие перспективные подходы, как легкие вычислительные потоки и тегированная общая память.

Ключевые слова: *экзамасштабный суперкомпьютер, модель параллельных вычислений с общей памятью, многоядерный процессор на ПЛИС*

Введение

Согласно представлению об архитектуре экзафлопсного компьютера (рис. 1) он строится из вычислительных модулей (ВМ), каждый из которых имеет один или несколько процессорных кристаллов с подсоединенными к ним блоками памяти и интерфейсами коммуникационной среды, объединяющей все ВМ [1, 2]. Процессорный кристалл содержит накристальную коммуникационную сеть (*interconnect*), объединяющую вычислительные и управляющие ядра, специальные вычислительные устройства, один или несколько блоков управления памятью (*Memory Management Unit*, *MMU*), блоки локальной памяти, включая кеш-памяти. Вычислительный модуль имеет также контроллеры блоков

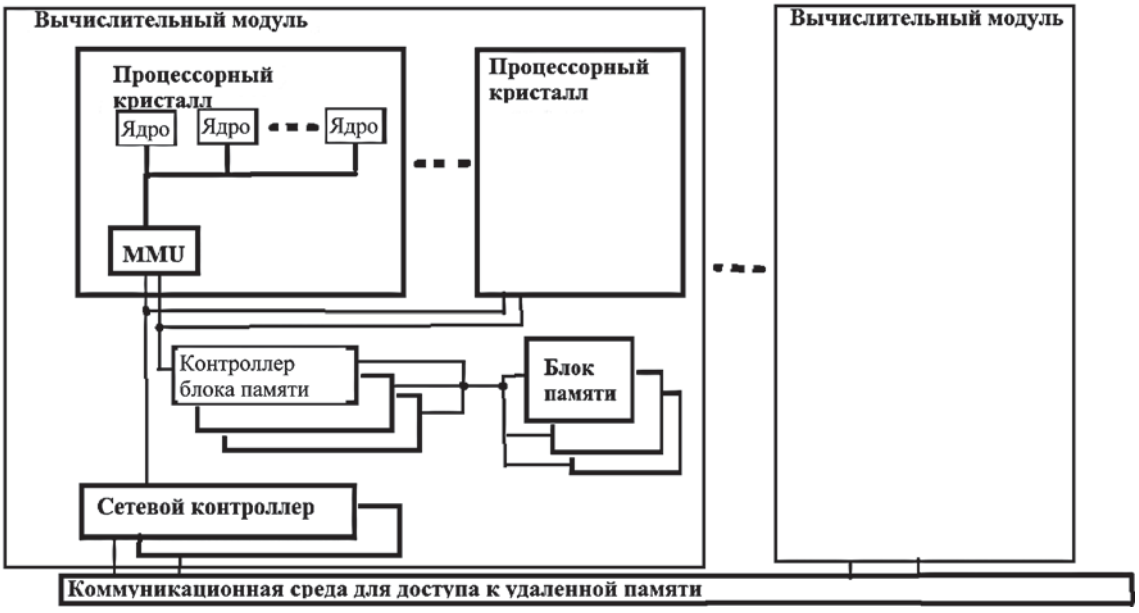


Рис. 1. Архитектура экзафлопсного суперкомпьютера

внешней памяти и один или несколько сетевых контроллеров.

Каждое ядро из числа размещаемых на кристалле при выполнении команды доступа к памяти обращается к накристальному блоку управления памятью. На основе задаваемого при инициализации распределения глобального адресного пространства по блокам памяти блок управления памятью определяет, куда идет обращение — к локальному или удаленному блоку памяти другого ВМ. В последнем случае формируется обращение к сетевому контроллеру, который должен переслать в другой ВМ обращение соответствующему удаленному блоку памяти. Таким образом, выполнение в ядре команды обращения к памяти задерживается в ожидании ответа о завершении обращения к памяти из MMU. Слова блоков памяти в целях реализации синхронизации обращений к общей памяти расширены специальным тегом, называемым FE-битом. Получив обращение из MMU, контроллер блока памяти выполняет работу с FE-битом выбранного слова памяти.

- Если FE-бит не имеет требуемого значения, то обращение к памяти помещается в таблицу ожидания. Строки этой таблицы содержат обращение к памяти, в состав которого входят: команда чтения/записи; адрес ячейки; значения FE-битов до выполнения обращения и по его завершении; указатель на незавершенную команду MMU.
- Если FE-бит имеет требуемое значение, то контроллер блока памяти выполняет требуемое чтение или запись и формирует заданное значение FE-бита. Не вдаваясь в детали реализации контроллера памяти, отметим, что после завершения обращения к ячейке памяти должен выполняться поиск в таблице ожидания строк, соответствующих этой ячейке памяти, а также проверка возможности выполнения соответствующего обращения к памяти. Если таковое возможно, то обращение пускается на выполнение и соответствующая ему строка удаляется из таблицы. Отметим, что описанные действия требуют ассоциативного поиска в таблице ожидания.

Существующие на настоящее время модели программирования на базе передачи сообщений не позволяют эффективно использовать потенциальный параллелизм алгоритмов обработки. Этот факт подтверждается статистикой применения компьютеров списка TOP 500. Создание других моделей экзафлопсного программирования необходимо, чтобы:

— эффективно использовать многократно возросший параллелизм обработки (10^9 и более потоков);

— при обеспечении высокой степени параллелизма не вносить дополнительных трудностей при разработке параллельных программ, т. е. не снижать продуктивности их разработки.

Более того, новые модели должны многократно в сравнении с существующим уровнем повышать эту продуктивность. Этому должно способствовать применение глобально адресуемой памяти, а также повышение уровня и непроцедурности средств параллельного программирования, которые характерны для новых моделей.

Для того чтобы обеспечить совместную разработку эффективной архитектуры целевой системы экзафлопного диапазона, ее программного обеспечения, методов и алгоритмов программирования, создается [3] моделирующая гетерогенная вычислительная система с использованием современных коммуникационных технологий и ВМ, реализованных на основе программируемых логических интегральных схем (ПЛИС). Таким образом, открывается возможность эмулирования архитектур, эффективно реализующих модели программирования без существенных затрат времени, свойственных имитационному моделированию.

В статье предложена модель экзафлопсного программирования и соответствующая этой модели реализация ВМ на ПЛИС, а также системное программное обеспечение. Такой подход позволяет создавать параллельные программы и оценивать их эффективность.

Модель и язык экзафлопсного программирования

В качестве основы модели экзафлопсного программирования выберем модель, известную как *Parallel Random Access Model* (PRAM) [4]. Эта модель базируется на архитектуре мультипроцессора с разделяемой памятью. Процессоры *Pi*, $i = 1, ..., n$, имеют доступ к общей памяти (*distributed shared memory*, DSM) или секционированной общей памяти (*partitioned global address space*, PGAS), которая физически распределена по ВМ (процессор + блок памяти), однако имеет общее адресное пространство и логически доступна всем процессорам. Создавая программу, пользователь предполагает, что команды потоков, протекающие в каждом из процессоров, выполняются синхронно, а межпроцессовые коммуникации и синхронизация осуществляются через ячейки памяти.

Возможны различные подходы к разрешению конфликтов доступа разных процессоров к одной ячейке разделяемой памяти [4]. В рамках нашего рассмотрения будем полагать приемлемым подход *concurrent-read exclusive-write* (CREW) с возможностью чтения одной и той же ячейки совокупностью процессоров, а записи только одним из процессоров. При этом все одновременные доступы к памяти по чтению предшествуют доступу по записи.

При такой модели параллельного программирования пользователь должен только указывать, какие вычисления можно проводить параллельно в соответствии с выбранным алгоритмом. Это способствует выявлению всего параллелизма, присущего алгоритму, и повышает продуктивность программирования.

При необходимости прочитать одно и то же значение многими потоками, а потом записать вместо него в соответствующую ячейку общей памяти новое значение, пользователь полагает, что при одновременном обращении к памяти команды чтения предшествуют команде записи. Такой механизм разрешения конфликта реализуется аппаратными средствами управления доступом к памяти.

В свое время именно отсутствие эффективного масштабируемого аппаратного решения для разрешения конфликтов доступа к памяти привело к отказу от PRAM и переходу к модели на базе передачи сообщений. Однако представляется, что такое решение на основе модели PRAM на настоящее время существует. Рассмотрим по порядку отдельно языковые средства порождения и завершения процессов и средства межпроцессовых коммуникаций и синхронизации в рамках предлагаемого расширения PRAM-модели.

Языком параллельного программирования, реализующим модель PRAM, может служить расширение языка C [4]. Для порождения и завершения асинхронных тредов введены три дополнительные функции: spawn (*a*, *n*); joint; fetch_and_add(*e*, *x*). Эти функции позволяют, соответственно: породить заданное параметром *n* число тредов с последовательными номерами, начиная с номера *a*; завершить тред, исполняющий join; выполнить как неделимую атомарную операцию присвоения переменной, например, номеру треда, значения параметра *x* с последующим увеличением значения *x* прибавлением к значению *x* значения *e*.

Исходя из необходимости реализации PRAM, ВМ должен выглядеть как "обычный" многопотоковый процессор, функционирующий под управлением операционной системы (ОС) семейства Unix. В таком модуле "нативно" реализованы алгоритмы распределения потоков, контроля загрузки ядер и т. д. Для "нативного" распараллеливания программы на множество потоков, фрагменты кода единой программы должны отправляться с очень низкими накладными расходами на множество исполнительных ядер. Ядра при этом получают указатель на фрагмент кода, передаваемого им для исполнения. Результаты сохраняются в общей памяти. В относительно устоявшейся терминологии это называется "подход на базе "легких потоков". Такой подход реализован, например, в библиотеке "легких тредов" *Qthreads* [5], созданной в Sandia National Laboratories. Основная идея — обеспечить механизм порождения параллельно исполняемых потоков на уровне процессора, а не на уровне ОС. В идеале накладные расходы на вызов такого потока сравнимы с затратами при вызове обычной функции. В рамках формализма *Qthread* порождение потока описывается вызовом следующего вида:

```
void start_thread(int(*)(int), int, ...);
```

Вызов может быть проведен из любого ядра. Исходной функции нужно передать указатель на запускаемую функцию, число аргументов и собственно аргументы, если такие имеются. Такой вызов совместно с атомарными, не блокируемыми операциями инкремента и некоторыми другими стандартными операциями создают возможность порождать (spawn) и объединять (join) группы параллельно выполняемых потоков, создавая последовательно-параллельную программу.

Синхронизация, которая выполняется на базе примитивов ОС, как в Unix-процессах и p-тредах [6], вызывает большие задержки. В рамках предлагаемой модели вычисления, проводимые каждым процессором, представляются потоком, в дальнейшем тредом. Для синхронизации тредов при этом предлагается использовать механизм межтредовой синхронизации и коммуникации. Его суть в добавлении к каждому слову памяти full/empty FE-бита и добавлении наряду с традиционными синхронизирующих команд обращения к памяти [7]. Команды writeef, readfe, readff и writeff, обращающиеся к ячейке памяти, могут выполняться только при определенном в них в первом компоненте суффикса значении FE-бита и оставляют после выполнения значение этого бита, заданное программистом во втором компоненте суффикса команды. Выполнение команды задерживается, если FE-бит не имеет требуемого значения. Например, команда writeff требует, чтобы перед ее выполнением значение FE-бита слова памяти, в которое будет запись, было full и оставляет после выполнения это же значение. Значение FE-бита full/empty обычно устанавливает, имеет ячейка памяти содержимое или нет.

Следует отметить, что в суперкомпьютере CrayXMT [7] расширение языка C для использования синхронизации на базе FE-битов реализовано как введение синхронизирующих переменных x$ и аппаратных функций (*generic functions*) для выполнения операций чтения и записи. Среди них отметим:

- purge x$ — присвоение FE-биту x$ значения empty;
- writeqr x$, *g* — запись в x$ значения *g*, если значение FE-бита x$ равно *q*, или ожидание записи, пока значение FE-бита не станет *q*; после записи значение FE-бита становится равным *r*;
- readqr x$ — чтение значения x$, если значение FE-бита x$ равно *q*, или ожидание чтения, пока значение FE-бита не станет *q*; после чтения значение FE-бита становится равным *r*.

Синхронизирующие переменные могут использоваться как семафоры для создания барьеров и критических интервалов, организующих исключительный доступ одного процесса из множества процессов к разделяемым этим множеством процессов переменным. Кроме этого, синхронизирующие переменные могут использоваться для задания потоковых (*dataflow*) вычислений. Например, вычисление $F(i, j) = 0{,}25(F(i-1, j) + F(i-1, j-1) + F(i, j-1) + +F(i, j)$ в области $0 < i < n+1$, $0 < j < n+1$ при заданных значениях $F(0, 0)$, $F(0, j)$, $F(i, 0)$ и установленных у $F(0, 0)$, $F(0, j)$, $F(i, 0)$ значениях FE-битов, равных full, может быть представлено следующим образом (символ $ используется для обозначения номера текущего процесса [4]):

```
spawn (1, n) {
int i;
i = $;
spawn (1, n) {
int j;
```

```
j = $;
purge (&(F[i, j]); //установка FE-битов, равных
//empty, в области 0 < i < n + 1, 0 < j < n + 1
}
}
spawn (1, n) {
int i;
i = $;
spawn (1, n) {
int j;
j = $;
double Left = readff(&(F[i, j - 1]);
double BottomLeft = readff(&(F[i - 1, j - 1]);
double Bottom = readff(&(F[i - 1, j]);
double t = readee(&(F[i, j]);
t = 0.25*(t + Left + BottomLeft + Bottom);
writeef(&(F[i, j]), t);
}
}
```

Программа порождает n тредов, каждый из которых порождает по n тредов. На первом шаге у ячеек памяти, хранящих $F(i, j)$ в области $0 < i < n + 1$, $0 < j < n + 1$, устанавливаются FE-биты, равные empty. Далее, аналогично предыдущему шагу, проводится порождение $n \times n$ тредов. В каждом треде вычисляется соответствующая функция. Сначала вычисляется $F[1, 1]$, так как исходно $F(0, 0)$, $F(0, 1)$, $F(1, 0)$ имеют FE-биты, равные full. Затем параллельно могут быть вычислены $F[2, 1]$ и $F[1, 2]$, так как появилось необходимое для их вычисления значение $F[1, 1]$ с FE-битом, равным full, затем параллельно могут быть вычислены $F[3, 1]$, $F[2, 2]$, $F[1, 3]$ и т. д.

Архитектура вычислительного модуля

Для экспериментального исследования представленной на рис. 1 архитектуры экзафлопсных компьютеров необходима вычислительная система с элементами, обеспечивающими возможность ее настройки на ту или иную проблемную область. Для такой системы должны быть достаточно легко адаптируемы востребованные и практически важные параллельные алгоритмы решения задач физики, химии, биологии и других областей применения. Должна экспериментально измеряться производительность таких параллельных программ и, главное, проводиться сравнение производительности реальных алгоритмов на вычислительных системах, в том числе перспективных и еще не реализованных на коммерчески доступных чипах архитектур.

Элементы конфигурируемой системы могут быть эмулируемы на традиционных вычислительных системах с использованием описания на языке аппаратных абстракций (*Hardware description language*, HDL). Следует однако отметить, что скорость такой эмуляции на настоящее время не позволяет смоделировать в реальном времени процессы, протекающие при решении какой-либо практически значимой задачи в целом. Вместе с тем любой перспективный язык программирования или подход к реализации программы на его основе может быть эмулирован на традиционных вычислительных системах. Скорость такой эмуляции обычно достаточна для проверки больших параллельных задач. Однако этот подход не позволяет оценить количественно рост производительности, связанный с реализацией перспективных аппаратных решений, так как по-существу использует традиционную элементную базу. Более интересным выбором, который, с одной стороны, достаточно производителен, а с другой стороны, позволяет учесть вклад тех или иных перспективных аппаратных решений, является создание прототипа на ПЛИС. При таком подходе реализуется эмуляция всего исследуемого аппаратного комплекса на наборе программируемых логических схем [3].

Рассмотрим прототип ВМ. С программной точки зрения он представляет собой иерархическую структуру из управляющих и вычислительных ядер, на каждом из которых возможен запуск треда прикладной программы. На уровне аппаратуры все ядра объединены сетью типа *wormhole*. Такая сеть обеспечивает передачу в пакетах различного типа и длины служебной информации, запросов к памяти, сообщений интерконнекта между ядрами. Архитектура ВМ представлена на рис. 2, см. вторую сторону обложки.

Вычислительный модуль на ПЛИС состоит из $N \times M$ предельно компактных универсальных вычислительных ядер Slave Cores, объединенных *wormhole*-сетью.

Вычислительное ядро построено на базе открытого компактного универсального 32-битного RISC-ядра MB-Lite, вся периферия разработана специально для настоящего проекта.

Каждое вычислительное ядро включает модуль контроллера шины Bus Controller (BC), позволяющий пропустить проходящее сообщение, вычитать его или добавить собственное сообщение в шину. Передача сообщений по шине реализована по принципу "отправил и забыл", в шине отсутствуют аппаратные подтверждения. Шина не требует для своей реализации больших затрат ресурсов, так как при передаче из одного узла в другой буферизуется только часть сообщения, а не все сообщение целиком. Передача сообщений происходит по алгоритму статической распределенной маршрутизации. Каждый пакет содержит заголовок, по которому коммутатор определяет, в какой из выходных портов отправить данное сообщение. Коммутация сигналов "3 в 3" позволяет обеспечить низколатентную передачу сообщений без дополнительных затрат ресурсов.

Логика шины работает на собственной частоте, которая обычно в 1,5—3 раза превышает тактовую частоту вычислительных ядер. Развязка частот шины и ядер позволяет: увеличить количество информации, передаваемой по шине за один процессорный такт; добиться передачи одного короткого сообщения за такт работы вычислительного ядра или быстрее. Около концов цепочек расположены буферы сообще-

ний Message Buffer (MB) для накопления сообщений в том случае, если расположенное за пределами шины устройство не в состоянии "на лету" обработать получаемые сообщения.

На одном конце каждой цепочки расположено управляющее ядро (Master Core), задача которого — управление загрузкой вычислительных ядер (Slave Core) и маршрутизация сообщений между различными цепочками. Для этого все Master Core объединены второй шиной, в которую включено головное ядро Super Master Core, на котором начинается выполнение любой программы. К Super Master Core подключены также основные периферийные устройства: Timer — таймер; UART — последовательный порт; DMA — модуль передачи блоков данных в память управляющего компьютера, если таковой присутствует в системе.

Выполнение любой параллельной задачи начинается с ее запуска на Super Master Core. В ходе исполнения программы возможны передачи управления на Slave Core. При этом удаленный запуск определяет только объем ресурсов, которые необходимы, конкретное распределение ресурсов динамически определяется сервисной программой, запущенной на всех Master Core.

Каждое вычислительное ядро обладает локальной кэш-памятью инструкций, в то же время каждая линейка содержит кэш инструкций второго уровня (Instruction Cache). Такой подход позволяет уменьшить время ожидания ответа на запрос инструкций из общей памяти, а также снизить нагрузку на контроллер памяти при запросе от многих Slave Core одних и тех же команд. Подобная ситуация является обычной, когда фрагмент кода запускается параллельно на многих ядрах цепочки.

Все взаимодействия вычислительного ядра с другими устройствами вне вычислительного ядра осуществляются с помощью сообщений, передаваемых по пакетной шине. Для отправки любого сообщения оно должно быть передано в контроллер шины BC. Пришедшие по шине сообщения, если они не являются считанной строкой кэша команд, также записываются в общую локальную память и могут быть прочитаны оттуда любым подключенным к ядру устройством. В зависимости от типа сообщений, их обработкой занимается один из четырех специализированных модулей. В зависимости от типа пришедшего сообщения контроллер шины сигнализирует соответствующему модулю о пришедших данных.

Для чтения/записи данных из/в глобальную память реализован модуль общения с общей памятью (*dmem*), который формирует запросы контроллеру общей памяти на запись и чтение, а также прием и обработку приходящих по шине данных. Для подкачки команд в локальный кэш команд (Instruction Cache) используется модуль обращения за строками кэша (*imem*). Наличие отдельной памяти позволяет уменьшить время простоя по шине команд процессорного ядра. Со стороны ядра реализован блочный доступ к общей памяти и к внутренней памяти других ядер, по сути аналогичный DMA в традиционных системах. За обработку данного вида запросов, а также отправку специальных запросов в контроллер общей памяти отвечает модуль DMM. В случае необходимости запустить вычислительный поток или передать данные другому ядру используется модуль обработки и отправки сообщений. Этот модуль записывает данные на отправку в локальную память и сигнализирует о готовности контроллеру шины.

Общая память системы доступна со всех ядер, подключена через модуль Memory Interface Generator (MIG) к контроллеру общей памяти и далее к шинному арбитру Bus switch. Арбитр осуществляет передачу запросов на чтение/запись от ядер в общую память и обратно. В каждый рабочий такт арбитр осуществляет передачу запроса от одной из цепочек, готовых отправить запрос в контроллер общей памяти. При передаче сообщений от контроллера памяти арбитр по номеру адресата определяет, в какую цепочку отправить данное сообщение. Запрос, пришедший от ядра, приводит к генерации соответствующего вида запросов в очередь в память. Ответ от памяти после обработки направляется либо в ядро, либо остается в контроллере для последующей обработки.

Контроллер памяти для каждого запроса независимо может работать в обычном режиме (запросы на чтение и запись выполняются в порядке поступления, не используют дополнительные признаки) или в расширенном режиме с учетом механизма FE-битов. Такой бит сопровождает в памяти каждое слово.

FE-бит = ‘1’ (empty) означает, что ячейка пуста, и данные недоступны для считывания, все запросы на чтение помещаются в очередь и ждут появления данных в ячейке, т. е. значения ‘0’ (full), тогда данные можно прочитать, при этом FE-биту присваивается значение empty. Далее никакой другой запрос на чтение этой ячейки не будет выполнен до тех пор, пока не будет выполнена разблокирующая операция записи, которая вновь "взведет" FE-бит в состояние full.

Созданный проект реализации в ПЛИС ВМ может использоваться как для генерации ВМ с различным числом $N \times M$ универсальных ядер, так и ВМ, в которых к универсальным ядрам подключены спецустройства. Такими устройствами могут быть специализированные математические сопроцессоры для работы с векторами и матрицами, в том числе с плавающей точкой с двойной и более точностью, специальные модули для решения задач дискретной математики. Таким образом, можно создавать конфигурации аппаратных средств, эффективные для конкретных вычислительных задач.

Реализация вычислительного модуля в ПЛИС

Для практической реализации (см. таблицу) были выбраны три платформы: Xilinx ML-505 (Digilent Genesys), Rosta RSP-527, Rosta RC-47. На них удалось

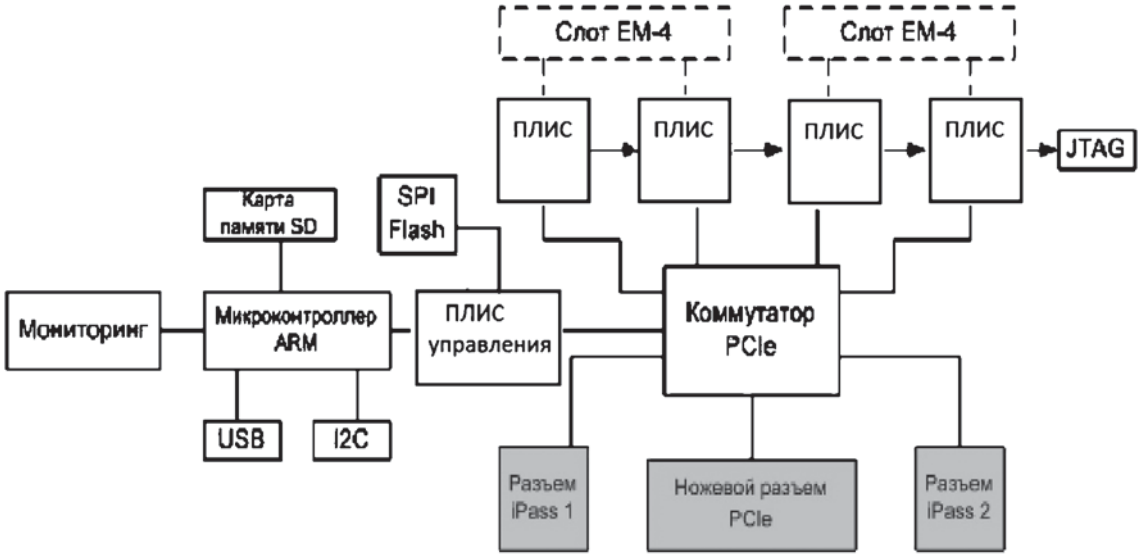
Характеристики платформ

Плата	Модель ПЛИС	Число ПЛИС на плате	Память (на 1 ПЛИС)	Интерфейс	Число ядер системы (без сопроцессоров)
Xilinx ML-505 (Digilent Genesys)	Virtex-5 X5VLX50T	1	256 Мбайт DDR2	Ethernet	6
Rosta RSP-527	Virtex-6 XC6VLX240T	2	16 Мбайт ZBT SRAM	PCIe	80
Rosta RC-47	Virtex-7 XC7VL2000T	4	1 Гбайт RLDRAM3	PCIe	1024

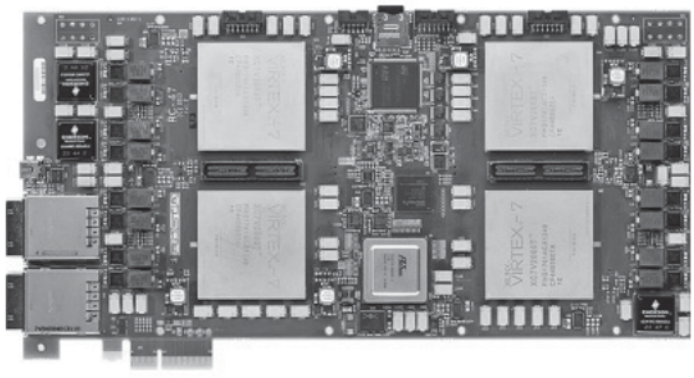
разместить 6, 80 и 1024 универсальных вычислительных ядер соответственно.

Основной на настоящее время является платформа Rosta RC-47 на базе четырех ПЛИС Xilinx Virtex-7 2000T, структура (*а*) и внешний вид (*б*) которой показаны на рис. 3.

Вычислительное устройство RC-47 — кластер из четырех ПЛИС Virtex-7 — третье поколение устройств, разработанных ООО НПО "Роста". Устройство RC-47 представляет собой PCI-express плату размером 150×280 мм, на которой установлены четыре ПЛИС Xilinx XC7V2000T, соединенные между собой при помощи коммутатора PEX8732 шиной PCI Express. Два кабельных разъема КС1 и КС2 шины PCI Express предназначены для масштабирования устройства.



а)



б)

Рис. 3. RC-47 — вычислительная плата с четырьмя ПЛИС Virtex-7, соединенными интерфейсом PCI Express:
а — структура; *б* — внешний вид

Для обеспечения надежного функционирования устройства предназначена система жизнеобеспечения LSS (*Life Support System*). Эта система построена на базе микроконтроллера STM и ПЛИС Spartan-6. Она позволяет управлять различными параметрами платы, проводить мониторинг ее состояния.

На плате реализованы два посадочных места для установки мезонинного модуля расширения в формате EM Slot type 4. При установке на эти посадочные места модулей памяти типа EM4-DDR3 к каждой ПЛИС подсоединяется внешняя память объема до 1 Гбайта.

Программное обеспечение вычислительного модуля

Разработан образец комплекта программного обеспечения проекта формирования в ПЛИС ВМ. В том числе разработано: руководство по сборке и запуску проекта; руководство по запуску проекта на нескольких ПЛИС. Структурированы исходные коды проекта, разработаны и протестированы VHDL-описания всех модулей проекта. Для всех типов используемой памяти разработаны карты памяти и процедуры инициализации. Разработана библиотека Libmalt, содержащая функции, отражающие специфику архитектуры вычислителя. Описаны интерфейсы всех модулей, использование которых может быть необходимо сторонним разработчикам. Разработано программное обеспечение для поддержки виртуальных последовательных портов поверх шины PCIe, программное обеспечение для работы с PCIe-устройствами посредством DMA.

Отметим, что столь же важным, как выбор языка программирования, является выбор ОС. Аппаратные платформы, не имеющие собственной ОС, в настоящем не имеют шанса найти и занять свою нишу в области обработки данных. Причина в том, что множество написанных и создаваемых программ подразумевают наличие ОС и эти программы просто не могут ни при какой адаптации быть запущены на "голом железе". Причем требуется не просто ОС, а POSIX-совместимая ОС, сборка на которой существующих прикладных пакетов может быть проведена без существенных переработок последних.

Однако распространенные ОС либо закрыты правообладателями и не допускают модификаций (Windows), либо достаточно сложны (Linux) и требуют значительной переработки для их использования в составе существенно многоядерных систем. Однако есть исключения, всем перечисленным выше требованиям удовлетворяет Minix — свободная Unix-подобная микроядерная ОС, распространяемая по лицензии BSD. Исходные коды Minix исключительно лаконичны и хорошо документированы. Развиваемая с 2005 г. ОС Minix3 доведена до возможности ее использования в качестве надежной ОС. Она компактна по объему кода, а коммуникации между драйверами, серверами и прикладными программами выполнены так, что могут быть нативно перенесены на системы с сотнями и тысячами ядер. По этим причинам в представленном проекте вычислительной системы Minix3 была выбрана в качестве создаваемого прототипа.

Заключение

В данной работе описана модель вычислений и организации памяти для экзамасштабных суперкомпьютеров, представлен действующий прототип узла суперкомпьютера экзафлопсного уровня, реализованный на ПЛИС. Этот прототип содержит более тысячи вычислительных ядер и предлагает пользователю при решении практических параллельных задач использовать такие перспективные подходы, как легкие вычислительные потоки и тегированная общая память.

Созданный в ходе выполнения проекта прототип ВМ в ПЛИС может использоваться как для генерации ВМ с различным числом $N \times M$ универсальных ядер, так и ВМ, в которых к универсальным ядрам на уровне регистров подключены спецустройства. В последнем случае проекты реализации в ПЛИС этих спецустройств должны быть добавлены к базовому проекту ВМ. Таким образом, предоставляется возможность создавать конфигурации аппаратных средств, оптимальные для программируемого алгоритма обработки.

Для всех получаемых конфигураций аппаратных средств может быть использовано созданное программное обеспечение и технология параллельного программирования, реализующая модель разделяемой памяти.

Список литературы

1. **Корнеев В. В.** Подход к программированию суперкомпьютеров на базе многоядерных мультитредовых кристаллов // Вычислительные методы и программирование. 2009. Т. 10. С. 123—128.

2. **Корнеев В. В.** Модель программирования и архитектура экзафлопсного суперкомпьютера // Открытые системы. 2014. № 10. С. 20—22.

3. **Горбунов В. С., Елизаров С. Г., Корнеев В. В., Лацис А. О.** Футурология суперкомпьютеров // Суперкомпьютеры. 2014. № 17. С. 24—28.

4. **Wen X., Vishkin U.** FPGA-Based Prototype of a PRAM-On-ChipProcessor // Proceedings of the 5th conference on Computing frontiers. May 5—7, 2008, Ischia, Italy. New York, NY, USA: ACM, 2008. P. 55—66.

5. **Wheeler K., Murphy R., Thain D.** Qthreads: An API for Programming with Millions of Lightweight Threads // In Proceedings of the 22nd IEEE International Parallel & Distributed Processing Symposium. April 2008. IEEE Computer Society Press, 2008. P. 1—8.

6. **IEEE** Std 1003.1—1990: Portable Operating Systems Interface (POSIX.1). Institute of Electrical and Electronics Engineers. 1990.

7. **Cray** XMT™ Cray XMT™ Programming Model. S—2367—201.

S. G. Elizarov, Senior Staff Scientist, M.V. Lomonosov Moscow State University,
G. A. Lukyanchenko, Postgraduate Student, National Research Centre "Kurchatov Institute", Moscow,
V. V. Korneev, Professor, Director for Research, e-mail: korv@rdi-kvant.ru,
State Enterprise "Research and Development Institute "Kvant", Moscow

Exascale Computer Parallel Programming Technology

Parallel programming model based on PRAM — Parallel Random Access Model and memory architecture for exascale supercomputer are proposed. Tagged shared memory with additional full/empty bits for each word that keeps the reading from an empty compartment till the data will be written on it is used for preventing memory access conflicts and data synchronization purposes. For native parallel programming we use the 'lightweight threads' approach dividing the program into multiple threads performing relatively short code fragments from a single program with very low overheads. The results are saved in the shared memory. Are described the main programming language features necessary for the offered model.

We present the prototype of FPGA-based exascale supercomputer node made by the model offered above. Node contains more than 1000 cores. Main multiprocessor units including universal cores computing kernels, on-chip communication network, advanced memory controllers etc. are described. Hardware platforms characteristics are provided. Main tests were done on Rosta RC-47 test bench containing four largest Virtex7 FPGAs, eight RLDRAMIII memory channels and PCI Express Gen3 controller. Node prototype was running with Minix3 OS.

The prototype offers a very flexible architecture including control of power and quantity of universal cores and custom computational blocks. User can tune throughput and latency of the memory controllers, communication network and many other blocks of prototype for it's particular tasks. So the exascale supercomputer node prototype offers the possibility of supercomputer's speed growth experimental research using such perspective approaches like a lightweight threads and tagged shared memory.

Keywords: *exascale supercomputer, shared memory parallel programming model, multicore processor on FPGA*

References

1. **Korneev V. V.** Podhod k programmirovaniyu superkompyuterov na baze mnogoyadernyh multitredovyh kristallov. *Vychislitelnye metody i programmirovanie*, 2009, vol. 10, pp. 123—128 (in Russian).

2. **Korneev V. V.** Model programmirovaniya i arhitektura ehkzaflopsnogo superkompyutera. *Otkrytye sistemy*, 2014, no. 10, pp. 20—22 (in Russian).

3. **Gorbunov V.S., Elizarov S.G., Korneev V.V., Lacis A.O.** Futurologiya superkompyuterov. *Superkompyutery*, 2014. no. 17, pp. 24—28 (in Russian).

4. **Wen X., Vishkin U.** FPGA-Based Prototype of a PRAM-On-ChipProcessor. *Proceedings of the 5th conference on Computing frontiers.* May 5—7, 2008, Ischia, Italy. New York, NY, USA: ACM, 2008, pp. 55—66.

5. **Wheeler K., Murphy R., Thain D.** Qthreads: An API for Programming with Millions of Lightweight Threads. *In Proceedings of the 22nd IEEE International Parallel & Distributed Processing Symposium*. April 2008. IEEE Computer Society Press, 2008, pp. 1—8.

6. **IEEE** Std 1003.1—1990: Portable Operating Systems Interface (POSIX.1). Institute of Electrical and Electronics Engineers. 1990.

7. **Cray** XMT^{TM} Cray XMT^{TM} Programming Model. S—2367—201.

ИНФОРМАЦИЯ

Международный конгресс
по интеллектуальным системам и информационным технологиям

3—9 сентября 2015 года,

Россия, Черноморское побережье, Геленджик-Дивноморское

Мероприятия, проводимые в рамках конгресса

- Международная научно-техническая конференция <Интеллектуальные системы AIS'15>
- Международная научно-техническая конференция <Информационные технологии IT'15>
- Международная научно-техническая конференция <Интеллектуальные САПР CAD-2015>
- Молодежная научно-техническая конференция <Информационные системы и технологии-2015>
- Международный семинар <FRUCT>
- Круглые столы по проблемам интеллектуальных систем и информационных технологий
- Ярмарка-выставка программных продуктов

Официальный сайт конгресса http://icai.tti.sfedu.ru/

УДК 004.05

В. Е. Гвоздев[1], д-р техн. наук, проф., зав. каф., e-mail: wega55@mail.ru,
О. Я. Бежаева[1], канд. техн. наук, доц., e-mail: obezhaeva@gmail.com,
Р. Р. Курунова[1, 2], аспирант, инженер, e-mail: roksana.kurunova@gmail.com,
[1] Уфимский государственный авиационный технический университет,
[2] ОАО "НИИ "Солитон", г. Уфа

Выявление противоречий в требованиях к программному продукту на основе исследования непрямых связей между ними*

Конечной целью исследований, результаты которых представлены в статье, является разработка формальных процедур установления неявных противоречий в системах требований к программному продукту. Предлагаемые подходы ориентированы на решение задачи обеспечения одного из базовых свойств требований к программным продуктам — непротиворечивости. Рассмотрены подходы к анализу характера и силы транзитивных связей между требованиями. Основу решения задачи составляет системное сочетание технологии QFD-проектирования, использование аппаратов знаково-ориентированных и нечетких графов. Предложена нелинейная структурная модель "каскад домов качества", отличающаяся от известной линейной модели тем, что в ней отражены транзитивные связи между требованиями внутри дома качества, в также внешние обратные связи между домами качества. Предлагаемые подходы позволяют, во-первых, адаптировать аппарат QFD, широко применяемый в различных инженерных приложениях, в область анализа проектных решений, связанных с IT. Во-вторых, они позволяют разработать формальные процедуры выявления скрытых противоречий между требованиями, что повышает качество управления требованиями.

Ключевые слова: *непротиворечивость требований к программным продуктам, транзитивные связи, противоречия между требованиями, QFD, дом качества, знаково-ориентированные графы, нечеткие графы, анализ проектных решений*

Введение

Противоречия присущи большинству задач, требующих решения, в том числе задачам, связанным с формированием требований, которые предъявляются к программным продуктам на разных стадиях их жизненного цикла. Причинами возникновения таких противоречий являются:

- разное видение различными субъектами управления потребительских свойств программного продукта [1];
- различное оценивание проблемных ситуаций разными субъектами управления [2];
- неопределенность состояния внешней по отношению к объекту управления среды [3, 4] и внутренней среды объекта управления [4, 5], а также уникального сочетания внешней и внутренней сред [6];
- отказ заинтересованных сторон от ранее данных обещаний и нарушение в одностороннем порядке достигнутых договоренностей [4, 7];
- нелинейная природа сложных систем, вследствие чего незначительные административные и технологические ошибки могут в итоге оказать серьезное негативное влияние на свойства объекта управления и ход выполнения проекта [4, 8].

Раннее выявление противоречий является критическим фактором успеха управления сложными системами. Своевременное выявление противоречий и нахождение способов их разрешения предотвращает появление трудностей, производных от этих противоречий, и в итоге позволяет сократить затраты на создание изделия.

Признание программных систем разновидностью сложных систем на конгрессе Международной ор-

* Работа выполнена при поддержке гранта РФФИ № 14-08-97036.

ганизации по обработке информации IFIP в 1965 г. обосновало возможность адаптации методологий и полезных практик, хорошо зарекомендовавших себя при исследовании сложных систем других классов, в область программной инженерии. Вместе с тем, например, в работе [9] отмечается, что "...многие достижения в области теории и практики управления качеством сложной промышленной продукции, как правило, неизвестны и не используются специалистами, создающими и применяющими системы на базе программных средств". В той же работе подчеркивается, что оценка свойств программных продуктов на разных стадиях жизненного цикла зависит от интуиции и квалификации разработчиков, заказчиков, пользователей. Это является одной из причин недостаточной функциональной пригодности программных продуктов и неудовлетворительной эффективности программных проектов. Таким образом, анализ возможности и разработка способов адаптации известных и хорошо зарекомендовавших себя практических способов (практик) создания, использования, модификации и развития сложных промышленных систем в область программной инженерии следует отнести к факторам управления качеством сложных программных продуктов.

Качество требований к потребительским свойствам программного продукта относится к ключевым факторам, определяющим функциональную пригодность изделия [5, 10, 11]. К настоящему времени разработаны руководства [12], стандарты и подобные документы, в которых определен перечень признаков, которыми должны обладать "хорошие" спецификации требований к программным продуктам. Вместе с тем не определены четкие критерии оценки соответствия этим признакам, а также описание механизмов, использование которых обеспечивает формирование "хороших" спецификаций.

Формирование взаимосвязанных требований к потребительским свойствам создаваемого продукта, особенно на начальных стадиях жизненного цикла таких продуктов (изделий), относится к классу слабоструктурированных задач [1, 5, 9, 10]. Дефекты в требованиях являются одной из основных причин нарушения базового опорного плана программного проекта. Статья посвящена вопросу адаптации хорошо зарекомендовавшей себя и широко используемой при создании промышленных изделий технологии проектирования требований к свойствам объектов на разных стадиях их жизненного цикла. Основу адаптации составляет новая идея совместного использования модели дома качества, аппарата знаково-ориентированных графов и аппарата нечетких когнитивных карт. Выбор такого сочетания известных методов обусловлен следующими причинами.

- Формальная модель дома качества позволяет, во-первых, в компактной форме одновременно описать связи к требованиям потребительских свойств изделия с требованиями к конструктивным характеристикам изделия; во-вторых, описать связи между требованиями к потребительским свойствам изделия, в-третьих, описать связи между требованиями к конструктивным характеристикам изделия, в-четвертых, обеспечить сопоставление требований к потребительским свойствам и конструктивным характеристиками создаваемого изделия с аналогичными свойствами и характеристиками уже существующих изделий (выполнить внешний и внутренний бенчмаркинг [13]).
- Аппарат знаковых ориентированных графов и аппарат нечетких когнитивных карт являются эффективными инструментами описания слабоструктурированных ситуаций, а также отображения персональных знаний экспертов. Такие особенности моделей имеют особую ценность на ранних стадиях жизненного цикла программных продуктов. Для этих этапов характерна высокая неопределенность свойств программного продукта.

1. Применение QFD в инженерных дисциплинах

Одним из инструментальных средств разноаспектного описания свойств сложных систем является *Quality Function Deployment* — QFD. Целью QFD является уменьшение неопределенности при решении проектных задач на разных стадиях жизненного цикла изделий за счет реализации процедуры системного описания слабоструктурированных объектов и ситуаций. В работах [14, 15] и ряде других приводятся сведения и описания, которые позволяют дать следующие особенности, определяющие QFD.

- QFD — это структурированный процесс, язык визуального моделирования, последовательность внутренне связанных диаграмм, ориентированных на решение проектных задач, характерных для разных стадий жизненного цикла изделий.
- QFD — это инструментарий заказчика, который позволяет встроить его требования в проектную документацию. Цель этого инструментального средства — убедиться, что требования заказчика интегрированы в каждую часть проекта, от определения границ проекта через этап его планирования до процесса контроля за ходом выполнения проекта и его закрытия. Функция качества позволяет выявить требования заказчика и перевести их на язык проекта.
- QFD — это кросс-функциональный инструментарий (*cross-functional tool* [16]), обеспечивающий необходимую глубину понимания инженерами потребностей потребителей и создающий возможность формирования на их основе ранжированных по значимости с точки зрения потребителя требований, корректных с технических позиций.

В работе [17] отмечено, что QFD относится к методам, направленным одновременно на обеспечение высокого качества изделий и сокращения стоимости их создания. Инструментарий QFD позволяет реализовать удобные и высокопроизводительные механизмы системного проектирования, использование которых создает предпосылки того, чтобы из поля

зрения не выпали вопросы, важные с точки зрения потребителей.

Во многих литературных источниках, посвященных описанию QFD, основное внимание уделяется построению "каскадов домов качества". Иными словами — построению линейной архитектуры (понятие линейной и нелинейной архитектуры расшифровано в работе [18]), позволяющей поэтапно трансформировать "голос потребителя" (*voice of customer*) в характеристики системы управления производством изделия. Как правило, описывается каскад из четырех домов качества: "требования потребителя — потребительские свойства продукта"; "потребительские свойства продукта — измеримые характеристики продукта"; "измеримые характеристики продукта — характеристики процесса производства"; "характеристики процесса производства — характеристики системы управления производством".

В работе [19] приведено описание восьми домов качества. Дополнительно к упомянутым обсуждаются следующие структуры: "требование потребителя — требование потребителя" (так называемое "крыльцо дома качества"); "характеристики качества — характеристики качества" ("крыша дома качества"); "требования пользователя — функции"; "причины отказов — функции". В той же работе приведена ссылка на работу [20], в которой дается описание тридцати домов качества.

Относительно области применения QFD в литературе отмечается, что методология проектирования QFD позволяет получить подтверждение выполнения явных (обязательных) требований и определяет точки возможного достижения "возбуждающих" требований. Возбуждающими будем именовать требования, превосходящие ожидания потребителей.

В работе [17] отмечено, что QFD основана на водопадной модели жизненного цикла проекта, включающей такие виды деятельности, как концептуальное проектирование; проектирование проекта; создание прототипа; испытание и оценка; подготовка проектирования; развертывание проекта; реализация проекта; поддержка процесса проектирования; развитие процессов проектирования; завершение проекта.

В работе [21] подчеркнуто, что известные инструментальные средства проектирования (SADT, QFD, RCA) являются эффективными в описании и анализе вопросов и затруднений, возникающих в ходе решения проектных задач. Однако они имеют ограниченные возможности как средства, способствующие генерации новых идей и избегания ловушек, возникающих в ходе проектирования изделий. Это объясняется ограниченными возможностями преодоления психологической инерции, обусловленной предыдущим жизненным опытом проектировщиков.

Дом качества может рассматриваться как разновидность формальной когнитивной карты (ФКК) [22]. Когнитивная карта представляет собой причинно-следственные связи между понятиями [23]. Дому качества присущи все формальные признаки ФКК:

- прямоугольная и треугольная матрицы могут быть преобразованы в ориентированный граф, вершинам которого соответствуют компоненты "комнат" дома качества;
- дуги графа могут рассматриваться как прямые причинные влияния между узлами графа с указанием силы и знака влияния.

Формальные когнитивные карты или карты с формальной семантикой позволяют применять для поиска и принятия решений по управлению сложными и слабоструктурированными ситуациями формальные методы [24].

2. Анализ требований на основе QFD и знаково-ориентированных и нечетких графов

Покажем вначале, что дом качества относится к технологиям описания слабоструктурированных ситуаций. Во-первых, "крыльцо" и "крыша" дома качества представляют собой системы. Это подтверждается наличием треугольных матриц, описывающих связи компонентов "крыльца" и "крыши" (требований к программному продукту, соответствующим разным стадиям его жизненного цикла). Во-вторых, эти системы связаны между собой. Это подтверждается наличием матрицы взаимосвязей компонентов "крыльца" и "крыши". В-третьих, составы компонентов "крыльца" и "крыши" определяются в результате дискурса правообладателей [2], вовлеченных в реализацию программного продукта. Каждый из правообладателей определяет требования и характеристики программного продукта, исходя из своего видения проблемной ситуации (программный продукт рассматривается как инструментарий информационной поддержки урегулирования проблемной ситуации). Иными словами, альтернативные варианты домов качества возникают и анализируются в ходе коммуникативного процесса правообладателей. В-четвертых, характеристики связи параметров дома качества (после того как они будут выделены) являются не количественными, а качественными (лингвистическими оценками), образующими линейную упорядоченную шкалу. Характеристики связи определяются не посредством объективных измерений, а в результате дискурса правообладателей, т. е. являются субъективными оценками экспертов.

Из изложенного выше можно заключить, что дом качества является инструментарием описания слабоструктурированных ситуаций. Можно также утверждать, что дом качества — это модель представления знаний экспертов о проблемной ситуации, которая служит основой определения компонентов "крыльца", "крыши" и взаимосвязей между этими компонентами. Отсюда следует, что для исследования свойств дома качества можно использовать технологии когнитивных карт, в частности, знаково-ориентированных и нечетких графов.

Предположим, что дом качества для некоторого программного продукта имеет вид, представленный на рис. 1. Для упрощения материала матрица связи компонентов "крыльца" опущена. В ячейках матриц

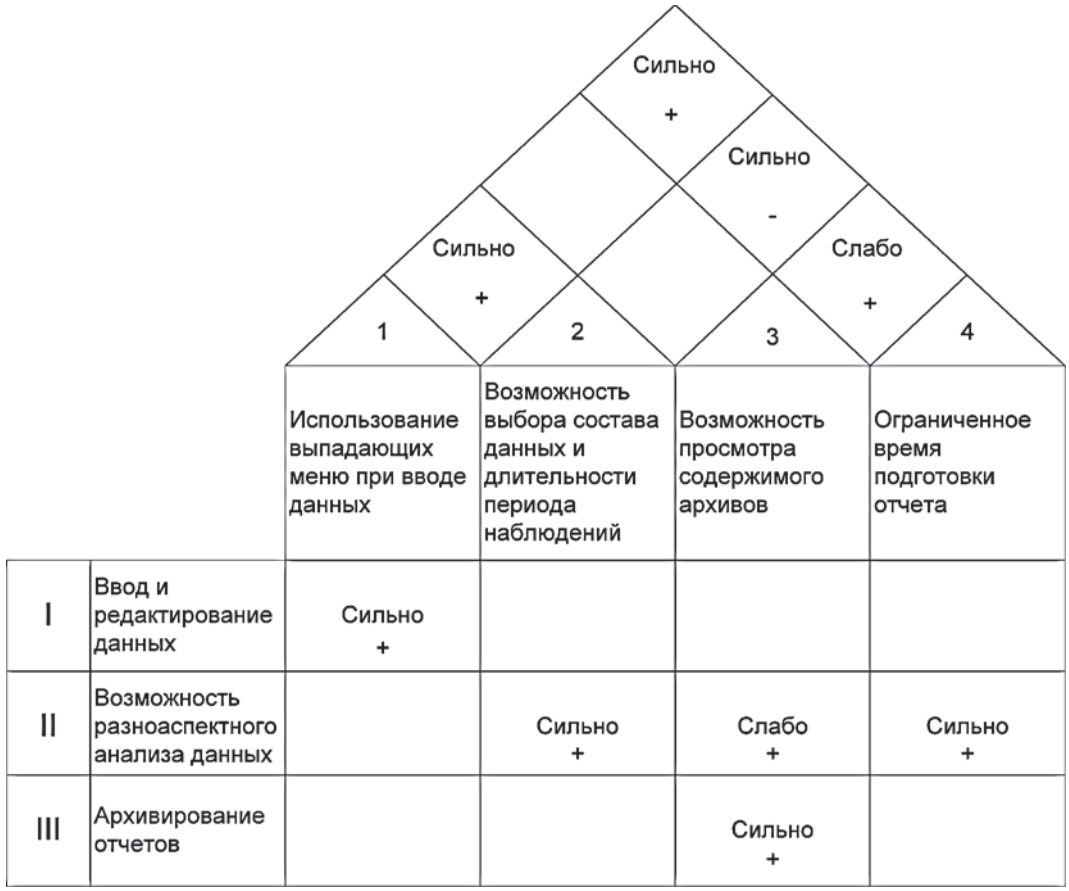


Рис. 1. Пример дома качества

Таблица 1

Содержание связей между компонентами "крыльца" и "крыши" дома качества

Признаки связей	Содержание связей	Характеристики связей
I—1	Наличие выпадающих меню позволяет сократить время ввода данных и уменьшить число ошибок при вводе. Чем более развита система меню, тем меньше время подготовки данных и выше качество данных	Сильно +
II—2	Возможность выполнения разноаспектного анализа данных за ограниченное время требует упрощения для пользователя процедуры определения состава данных и выбора длительности периода наблюдения	Сильно +
II—3	Возможность гибкого формирования запросов к содержимому архивов является необходимым признаком разноаспектного анализа данных. Это делает необходимым разработку интеллектуального интерфейса для работы с архивами	Слабо +
II—4	Требуемая глубина разноаспектного анализа при выбранных технологиях хранения, доступа и обработки данных накладывает ограничения на минимальное время подготовки отчета	Сильно +
III—3	Для анализа содержимого архивов требуется разработка интеллектуального интерфейса пользователей. Чем больше размер и размерность архива, тем более развитым должен быть инструментарий анализа его содержимого	Сильно +

Таблица 2

Содержание связей между компонентами "крыши" дома качества

Признаки связей	Содержание связей	Характеристики связей
1—2	Использование выпадающих меню позволяет упростить определение наборов данных, включаемых в отчет, и указание периода времени, за который нужно учитывать данные	Сильно +
1—4	Чем более развита система меню, тем проще сформировать задание на обработку и, следовательно, тем меньше время подготовки отчета	Сильно +
2—4	Чем больше объем данных и длительнее период наблюдения, тем больше время подготовки отчета	Сильно −
3—4	Возможность просмотра архивов готовых отчетов позволяет сократить время подготовки отчета, так как вместо дублирования текста можно сделать ссылку на ранее подготовленный отчет, в котором содержится необходимая информация	Слабо +

указаны знак и сила связи компонентов. В табл. 1 приведено описание содержания связей между компонентами "крыльца" и "крыши". В табл. 2 приведено описание содержания связей между компонентами "крыши" дома качества.

Предполагается, что степень связи, (сильно, слабо), а также знаки связей определены экспертами.

Анализ влияний на основе знаково-ориентированных графов

Рассмотрим подход к выявлению противоречий, основанный на аппарате знаково-ориентированных графов [25, 26]. Знаковый граф — это граф, дуги которого имеют веса +1 или −1, сокращенно обозначаемые "+" и "−". Знак "+" обозначает положительное влияние (рост фактора-причины приводит к росту фактора-следствия), знак "−" обозначает отрицательное влияние [27].

В рамках QFD связи между требованиями пользователей и характеристиками изделия являются неориентированными.

В основе анализа противоречий между разными требованиями лежит следующая формальная процедура.

1. На основе дома качества строится конечный помеченный связный граф, вершины которого разбиты на два подмножества W и W'. Элементы подмножества W соответствуют требованиям "крыльца" дома качества; элементы W' — требованиям "крыши".

2. Вершинам, принадлежащим W, присваиваются пометки v_0, ..., v_n, где n — число всех вершин, принадлежащих W.

3. На основе неориентированного графа строится симметрический граф [28, 29].

4. Последовательно перебирая все пары вершин v_i, v_j; $i, j = \overline{0,n}$; $i \neq j$, определяются все простые ориентированные пути, ведущие из v_i в v_j. Полагается, что хотя бы одна вершина простого пути принадлежит W'; в состав простого пути может входить не более двух вершин из W. Также полагается, что число вершин в простом пути должно быть не менее трех.

Рассмотрим влияние требования I на требование II (рис. 2).

Результаты анализа влияния приведены в табл. 3.

Знак, определяющий характер влияния вершины v_i на вершину v_j через p-й простой путь I_p, определяется соотношением

$$\operatorname{sign}\left(I_p\right) = \prod_{(k,l)\in E(p)} \operatorname{sign}\left(e_{kl}\right), \qquad (1)$$

где $E(p)$ — множество дуг простого пути I_p; $\operatorname{sign}(e_{kl})$ — знак дуги e_{kl}, соединяющей k-ю и l-ю вершины ($e_{kl} = (v_k, v_l)$), входящие в простой путь I_p. Если все простые пути из v_i *в* v_j имеют знак "+", считается, что i-е требование "крыльца" влияет на j-е требование "крыльца" положительно; если простые пути имеют знак "−" — влияние отрицательное; если части простых путей соответствует знак "+", а части простых путей — знак "−", характер влияния считается противоречивым. Если простые пути из v_i в v_j отсутствуют, считается, что i-е требование не влияет на j-е требование.

В силу того что различным I_p в рассматриваемом случае соответствуют разные знаки, можно за-

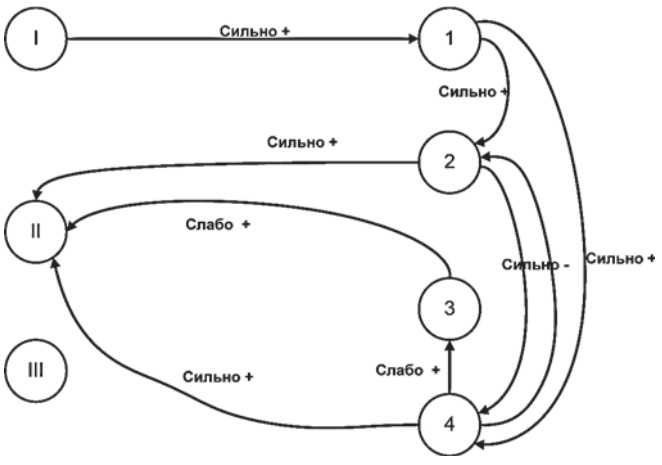


Рис. 2. Влияние требования I на требование II

Таблица 3

Простые пути и знаки влияния требования I на требование II

Простой путь в графе	Знак влияния
I → 1 → 4 → II	+
I → 1 → 2 → II	+
I → 1 → 2 → 4 → II	–
I → 1 → 4 → 2 → II	–
I → 1 → 4 → 3 → II	+
I → 1 → 2 → 4 → 3 → II	–

ключить, что итоговое влияние требования I на II противоречивое. Этот факт означает, что содержание анализируемых требований является источником потенциального противоречия. Действительно, с одной стороны, увеличение объемов данных создает условия для получения на их основе ранее неизвестной информации за счет разноаспектного анализа данных [30]. С другой стороны, увеличение объемов данных увеличивает затраты времени на подготовку (извлечение) данных и, как следствие, при заданных ограничениях — на срок предоставления отчета, сокращает время, отводимое на обработку данных.

Таблица 4

Простые пути и знаки влияния требования II на требование III

Простой путь в графе	Знак влияния
II → 4 → 3 → III	+
II → 2 → 4 → 3 → III	+
II → 3 → III	+
II → 2 → 4 → 3 → III	–

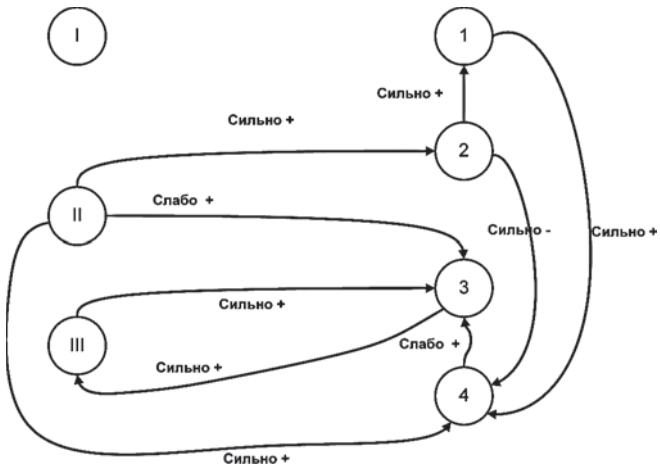


Рис. 3. Влияние требования II на требование III

Таблица 5

Простые пути и знаки влияния требования I на требование III

Простой путь в графе	Знак влияния
I → 1 → 4 → 3 → III	+
I → 1 → 2 → 4 → 3 → III	+
II → 3 → III	+
II → 2 → 4 → 3 → III	–

Таблица 6

Матрица взаимного влияния требований

Требование	I	II	III
I		Противоречивое	Нет противоречия Положительное
II	Противоречивое		Противоречивое
III	Нет противоречия Положительное	Противоречивое	

На рис. 3 представлено влияние требования II на требование III.

Результаты анализа влияния приведены в табл. 4.

Проведя по аналогичной схеме исследование всех возможных влияний остальных требований друг на друга (табл. 5), представим результаты в табл. 6.

Результаты анализа взаимного влияния требований позволяют на ранних стадиях проектирования выявить неявные противоречия между ними, что создает предпосылки для разработки мер по урегулированию противоречий. Возможным способом урегулирования являются компромиссные ограничения на основные характеристики проекта — бюджет и сроки реализации, а также на функциональные и нефункциональные характеристики программного продукта.

Анализ влияний требований посредством нечетких графов

Для анализа слабоструктурированных ситуаций с качественными и плохо определенными параметрами более адекватным является подход, основанный на понятии нечетких когнитивных карт (или нечетких графов) [27]. В общем случае нечеткий граф — это ориентированный граф. Его вершины представляют собой рассматриваемые элементы множеств, а дуги согласовываются с элементами отношений, т. е. это упорядоченные пары. Степень принадлежности каждой упорядоченной пары ассоциируется с каждой дугой как ее вес. Вес дуги определяется из

лингвистической шкалы и характеризует силу влияния одного элемента множества на другой.

В рассматриваемом примере поставим в соответствие каждой дуге лингвистическую переменную, характеризующую влияние по следующей шкале: сильно > средне > слабо.

Непрямое влияние inf(I_p) требований друг на друга через путь I_p вычисляется на основе модели Коско [27, 31]:

$$\inf\left(I_p\right) = \min_{(k,l)\in E(p)}\{w_{kl}\}, \quad (2)$$

где w_{kl} — вес дуги e_{kl} в пути I_p, принимающий значения в соответствии с определенной выше лингвистической шкалой, в пути I_p.

Суммарное влияние $T_{(i,j)}$ i-го требования пользователя на j-е требование определяется соотношением

$$T_{(i,j)} = \max_{p(i,j)}\left\{\inf\left(I_p\right)\right\}, \quad (3)$$

где максимум берется по всем простым путям p, существующим между i-м и j-м требованиями. Таким образом, операция $T_{(i,j)}$ выделяет наиболее сильное из непрямых влияний inf(I_P).

Обратимся к содержимому табл. 3.

Простому пути I → 1 → 4 → II соответствует следующая оценка:

inf($I_{(1)}$(I, II)) = min{сильно, сильно, сильно} = сильно.

Простому пути I → 1 → 2 → II соответствует следующая оценка:

inf($I_{(2)}$(I, II)) = min{сильно, сильно, сильно} = сильно.

Простому пути I → 1 → 2 → 4 → II соответствует следующая оценка:

inf($I_{(3)}$(I, II)) = min{сильно, сильно, сильно, сильно} = сильно.

Простому пути I → 1 → 4 → 2 → II соответствует следующая оценка:

inf($I_{(4)}$(I, II)) = min{сильно, сильно, сильно, сильно} = сильно.

Простому пути I → 1 → 4 → 3 → II соответствует следующая оценка:

inf($I_{(5)}$(I, II)) = min{сильно, сильно, слабо, слабо} = = слабо.

Простому пути I → 1 → 2 → 4 → 3 → II соответствует следующая оценка:

inf($I_{(6)}$(I, II)) = min{сильно, сильно, сильно, слабо, слабо} = слабо.

Оценка степени суммарного влияния:

$T_{(I, II)}$ = max{сильно, сильно, сильно, сильно, слабо, слабо} = сильно.

Проведя аналогичную обработку содержания таблиц влияния требования I на требование III, требования II на требование III, получим оценки взаимного влияния требований, представленные в табл. 7.

Системное сочетание технологий дома качества и технологий когнитивных карт позволяет разработать новые формальные методы выявления неявных противоречий между требованиями, составляющими "крыльцо" дома качества, а также оценить степени их взаимного влияния. Использование таких формальных методов создает основу для своевременной организации коммуникативного процесса между правообладателями в целях поиска компромиссных решений относительно основных характеристик программного проекта, а именно — бюджета и сроков реализации потребительских характеристик программного продукта и вытекающих из них.

Таблица 7

Оценки степени взаимного влияния требований посредством транзитивных связей

Требование	I	II	III
I		Сильно	Слабо
II	Сильно		Слабо
III	Слабо	Слабо	

Устранение противоречий может потребовать внесения изменений в структуры предшествующих домов качества. Установление дополнительных внутренних и внешних связей позволяет предложить модель каскада домов качества, отличную от модели, описанной во многих источниках, например, в работах [15, 19, 32].

На рис. 4 представлен каскад домов качества, отражающий развертывание требований к программной системе с учетом транзитивных связей между требованиями.

Совместное использование модели дома качества, аппарата знаковых ориентированных графов и аппарата нечетких когнитивных карт позволяет предложить новый формальный подход к решению задач, связанных с выявлением неявных взаимосвязей и оценкой степени взаимного влияния требований к потребительским свойствам изделия. Это в свою очередь позволяет формализовать исследование таких базовых свойств перечня требований, как непротиворечивость, модифицируемость, трассируемость [12].

3. Направления продолжения исследований

В настоящей статье не обсуждается вопрос совместного, явного и неявного влияния компонентов "крыльца" друг на друга. Явные влияния описываются матрицей, аналогичной по форме матрице "крыши" дома качества. Подобная структура описана, например, в работе [19].

В настоящей статье не затронут вопрос о том, как учесть итоговое неявное влияние компонентов "крыльца" по совокупности простых путей в графе. Для решения этой задачи может быть задействован, например, подход, основанный на использовании функции принадлежности лингвистической шкалы [31]. Заметим, что при использовании такого подхода исключается вопрос конфликта влияний, возникающий при использовании когнитивных карт Аксельрода.

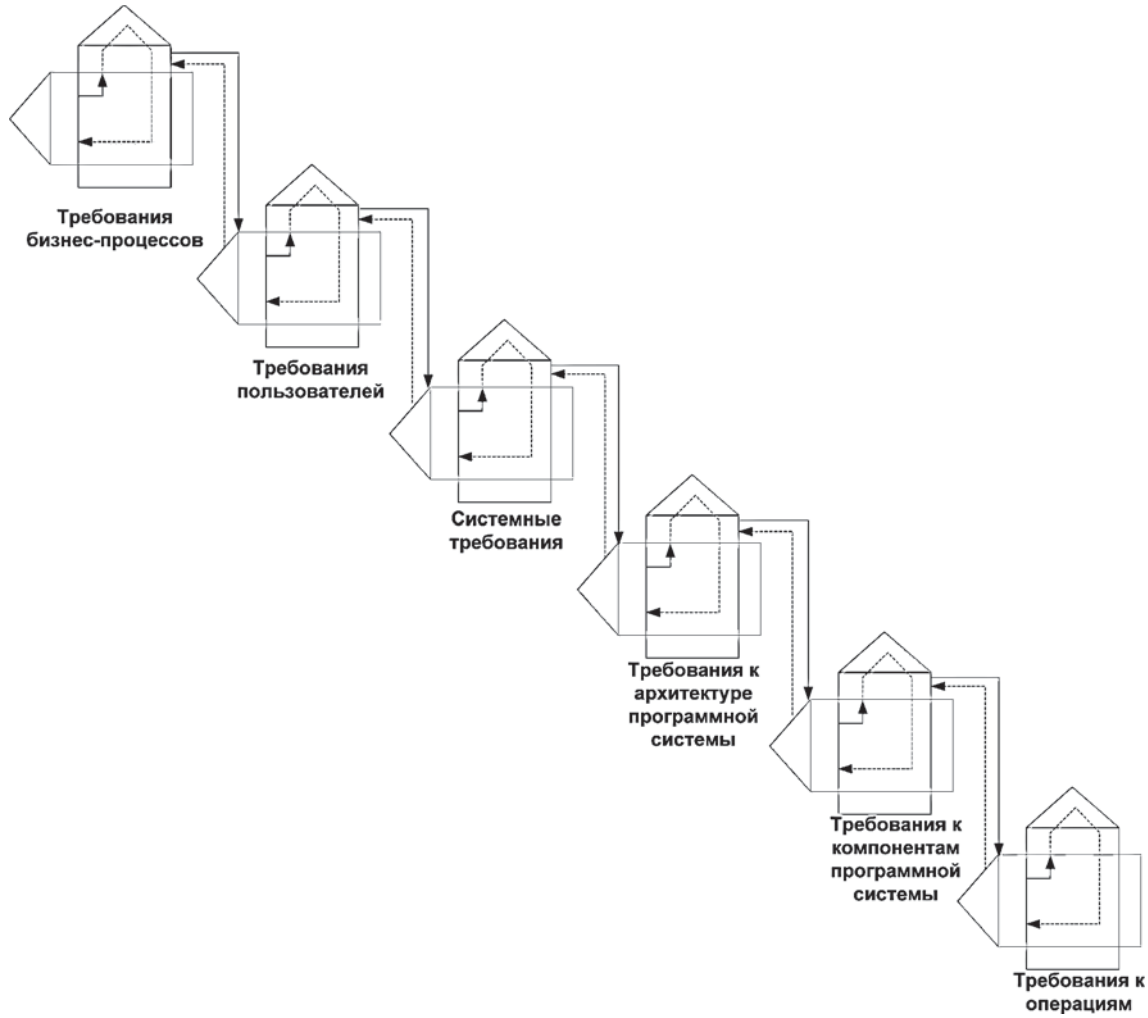


Рис. 4. Каскад домов качества с учетом внешних и внутренних обратных связей

В статье не затрагивается вопрос анализа ситуации, обусловленной противоположными оценками экспертов относительно степени и знака связи компонентов "крыльца" и "крыши" дома качества. Авторам представляется перспективным использовать в случае возникновения амбивалентных отношений между компонентами "крыши" и "крыльца" *NPN*-логики (*Negative-Positive-Neutral logic* [33]). Представляет также интерес не только анализ влияния изменений в требованиях на *k*-м этапе проекта на ранее принятые решения, но и оценка времени, в течение которого волна изменений достигнет каждый из ранее реализованных этапов. Важность разрешения вопроса, возникающего в силу влияния на код проекта того обстоятельства, что вносятся изменения в требования, обосновывается, например, в работе [7]. Известные методы трассировки требований основаны на использовании статических моделей. Авторам представляется, что возможной методической основой построения динамических моделей, соотнесенных с каскадом домов качества, окажется динамическая каузальная алгебра, предложенная в работе [34].

Заключение

Основу предполагаемого подхода к выявлению скрытых противоречий составляют базовые свойства "хороших" требований, определенные в работе [12]. Ограничениями предполагаемых подходов являются одинаковое и однозначное толкование содержания требований разными заинтересованными лицами, наличие единого мнения заинтересованных лиц относительно состава требований, соответствующих разным стадиям жизненного цикла программной системы, характеру и силе связей между требованиями. Иными словами, неопределенность проблемной ситуации, для решения которой создается программный продукт, такова, что можно определить состав бизнес-процессов, сформулировать соответствующие бизнес-процессам информационные задачи (хотя и не-

четкие), заинтересованным лицам сформулировать общую точку зрения (на концептуальном уровне) на проблемную ситуацию. Таким образом, предлагаемый подход ориентирован на ситуацию, когда центральным для принятия решений относительно программной системы является вопрос "как делать", а не "что делать".

Описанный подход к выявлению противоречий в требованиях, соответствующих разным стадиям жизненного цикла программной системы, следует рассматривать лишь как одно из инструментальных средств проектирования, дополняющее "ящик" средств, представленных, например, в работе [15].

Предлагаемые подходы позволяют разработать полностью формализованные процедуры выявления скрытых противоречий в требованиях, а также способы оценивания силы взаимовлияния требований. Их использование предоставляет возможность разработки программного инструментального средства для анализа качества требований на этапе построения моделей программных систем в форме спецификаций требований. Раннее обнаружение противоречий в спецификациях требований дает возможность своевременно вносить изменения в содержание требований и в конечном итоге позволяет сократить время и затраты на создание программных продуктов.

Список литературы

1. **Учебник** 4CIO. Версия 1.0. М.: 4CIO, 2011. 378 с.
2. **Виттих В. А.** Организация сложных систем. Самара: Самарский научный центр РАН, 2010. 66 с.
3. **Липаев В. В.** Анализ и сокращение рисков проектов сложных программных средств. М.: СИНТЕГ, 2005. 224 с.
4. **Rzevski G.** Managing Complexity. Проблемы управления и моделирования в сложных системах // Тр. XVI Междунар. конф., Самара, 2014. С. 3—12.
5. **Макконнел С.** Сколько стоит программный проект. М.: Русская редакция, СПб.: Питер, 2007. 297 с.
6. **Гвоздев В. Е., Ильясов Б. Г.** Пирамида программного проекта // Программная инженерия. 2011. № 1. С. 16—24.
7. **Йордон Э.** Путь камикадзе. М.: Лори, 2008. 290 с.
8. **Kaplan S., Visnepolschi S., Zlotin B., Zusman A.** New Tools for Failure and Risk Analysis. Ideation International Inc., 2005. 68 p.
9. **Липаев В. В.** Функциональная безопасность программных средств. М.: СИНТЕГ, 2004. 348 с.
10. **Липаев В. В.** Надежность программных средств. Серия "Информатизация России на пороге XXI века". М.: СИНТЕГ, 1998. 232 с.
11. **ГОСТ** 28195—89. Оценка качества программных средств. Общие положения.
12. **IEEE** Recommended Practice for Software Requirements Specifications IEEE STD 830—1998, June 1998.
13. **Каплан Р. С., Нортон Д. П.** Стратегические карты. Трансформация нематериальных активов в материальные результаты. М.: Олимп-Бизнес, 2005. 512 с.
14. **Shahin A.** Quality Function Deployment: A Comprehensive Review, 2005. 25 p.
15. **Милошевич Д.** Набор инструментов для управления проектами. М.: ДМК Пресс, 2008. 729 с.
16. **Mallon J. C., Mulligan D. E.** Quality Function Deployment — A system for meeting customers' needs // Journal of Construction Engineering and Management. 1993. Vol. 119, N. 3. P. 516—531.
17. **Dean E. B.** Quality Function Deployment: From the perspective of competitive advantage, 1998. URL: http://akao.larc.nasa.gov/dfc/qfd.html
18. **Саати Т.** Принятие решений. Метод анализа иерархий. М.: Радио и связь, 1993. 278 с.
19. **Bahill A. T., Chapman W. L.** A Tutorial on Quality Function Deployment // Engineering Management Journal. 1993. Vol. 5, N. 3. P. 24—35.
20. **King B.** Better Designs in Half the Time: Implementing Qfd Quality Function Deployment in America. Goal QPC Inc, 1989. 315 p.
21. **Kah-hin Chai, Jun Zhang, Kay-Chuan Tan.** National University of Singapore, ATRIZ-Based Method for New Service Design // Journal of Service Research. 2005. Vol. 8, N. 1. P. 48—66.
22. **Авдеева З. К.** Сравнительный анализ выборочных когнитивных карт по степени формализации // Когнитивный анализ и управление развитием ситуаций (CASC'2009): Тр. междунар. конф. М.: ИПУ РАН, 2009. С. 11—22.
23. **Vasantha Kandasamy W. B.** Fuzzy Cognitive Maps and Neutrosophic Cognitive Maps. Xiquan, 2003. 211 p.
24. **Абрамова Н. А.** Экспертная верификация при использовании формальных когнитивных карт. Подходы и практика // Управление большими системами. 2010. Т. 30.1. С. 371—410.
25. **Axelrod R.** The Structure of Decision: Cognitive Maps of Political Elites. Princeton University Press, 1976. 404 p.
26. **Робертс Ф. С.** Дискретные математические модели с приложениями к социальным, биологическим и экологическим задачам. Пер. с англ. М.: Наука, 1986. 496 с.
27. **Кузнецов О. П., Кулинич А. А., Марковский А. В.** Анализ влияний слабоструктурированными ситуациями на основе когнитивных карт // Человеческий фактор в управлении. М.: КомКнига, 2006. С. 313—345.
28. **Басакер Р., Саати Т.** Конечные графы и сети. М.: Наука, 1973. 368 с.
29. **Оре О.** Теория графов. М.: Наука, 1968. 352 с.
30. **Барсегян А. А., Куприянов М. С., Степаненко В. В., Холод Н. И.** Методы и модели анализа данных: OLAP и Data Mining. СПб.: БХВ—Петербург, 2004. 336 с.
31. **Pelaez E., Bowles J. W.** Using Fuzzy Cognitive Maps as a System Model for Failure Modes and Effects Analysis // Information Sciencies. 1996. Vol. 88. P. 177—199.
32. **Mazur G. H.** QFD for service Industries from Voice of Customer to Task Development // The Fifth Symposium on Quality Function Deployment Novi, Michigan June 1993, Japan Business Consultant, Ltd. 1993. P. 485—503.
33. **Zhang W. R., Chen S. S., Chen K. H. Zhang M., Bezdek J. C.** On NPN Logic // Proc. 18th IEEE Internat. Symp. MVL, Palma de Malorca, Spain, 1988. P. 381—388.
34. **Lin Z., Zhang J. Y.** Interrogating the structure of fuzzy cognitive maps // Soft Computing. 2003. Vol. 7. P. 148—153.

V. E. Gvozdev[1], Professor, e-mail: wega55@mail.ru,
O. Y. Bezhaeva[1], Associate Professor, e-mail: obezhaeva@gmail.com,
R. R. Kurunova[1, 2], Postgraduate Student, Engineer, e-mail: roksana.kurunova@gmail.com,
[1] Ufa State Aviation Technical University (USATU),
[2] Research Institute "Soliton", Ufa

Identification of Contradictions in the Requirements for Software Product Based on a Study of Indirect Links between them

The purpose of research is to develop formal procedures to establish the implicit contradictions in the systems of requirements to the software product. The proposed approaches oriented on the solution of providing one of the basic signs of the properties of requirements to software products — consistency. Examines approaches to the analysis of the character and strength of transitive relations between requirements, corresponding to different stages in the life cycle of software products. The basis of the solution of the problem is systemic combination of QFD, semantic-oriented and fuzzy graphs. Proposed nonlinear structural model of "cascade of houses of quality", which differs from the conventional linear model that in it reflects the transitive relations between the requirements of inside the house of quality, and also external backward linkages between the houses of quality. The proposed approaches allow, firstly, to adapt the QFD, widely used in various engineering applications, to the analysis of design decisions related to IT. Secondly, allow to develop formal procedures to identify the hidden conflict between the requirements, to improve the quality of management requirements.

Keyworlds: *consistency requirements for software products, transitive relation, the contradictions between the requirements, QFD, house of quality, semantic-oriented graphs, fuzzy graphs, analysis of design decisions*

References

1. ***Uchebnik** 4CIO. Ver. 1.0.* Moskow: 4CIO, 2011. 378 p. (in Russian).
2. **Vittih V. A.** *Organizacija slozhnyh sistem*. Samara: Samarskij nauchnyj centr RAN, 2010. 66 p. (in Russian).
3. **Lipaev V. V.** *Analiz i sokrashhenie riskov proektov slozhnyh programmnyh sredstv.* Moscow: SINTEG, 2005. 224 p. (in Russian).
4. **Rzevski G.** Managing Complexity. Problemy upravlenija i modelirovanija v slozhnyh sistemah. *Tr. XVI Mezhdunar. konf.* Samara, 2014. pp. 3—12 (in Russian).
5. **Makkonnel S.** *Skol'ko stoit programmnyj proekt*. M.: Russkaja redakcija, Saint Petersburg: Piter, 2007. 297 p. (in Russian).
6. **Gvozdev V. E., Il'jasov B. G.** Piramida programmnogo proekta. *Programmnaya ingeneria*, 2011. no. 1, pp. 16—24 (in Russian).
7. **Jordon Je.** *Put' kamikadze*. Moscow: Lori, 2008. 290 p. (in Russian).
8. **Kaplan S., Visnepolschi S., Zlotin B., Zusman A.** New Tools for Failure and Risk Analysis. Ideation International Inc., 2005. 68 p.
9. **Lipaev V. V.** *Funcionalnaya bezopasnost programmnyh sredstv.* Moscow: SINTEG, 2004. 348 p. (in Russian).
10. **Lipaev V. V.** *Nadejnost programmnyh sredstv.* Seriya "Informatizaciya Rossii na poroge XXI veka". Moscow: SINTEG, 1988. 232 p. (in Russian).
11. **GOST** 28195—89. Ocenka kachestva programmnyh sredsv. Obshie polojeniya (in Russian).
12. **IEEE** Recommended Practice for Software Requirements Specifications IEEE STD 830—1998, June 1998.
13. **Kaplan R. S., Norton D. P.** *Strategicheskie karty. Transformaciya nematerialnyh aktiviv v materialnye rezyltaty.* Moscow: ZAO "Olimp-Biznes", 2005. 512 p. (in Russian).
14. **Shahin A**. Quality Function Deployment: A Comprehensive Review. 2005. 25 p.
15. **Miloshevich D.** *Nabor instrumentov dlja upravlenija proektami.* Moscow: DMK Press, 2008. 729 p. (in Russian).
16. **Mallon J. C., Mulligan D. E.** Quality Function Deployment — A system for meeting customers' needs. *Journal of Construction Engineering and Management,* 1993, vol. 119, no 3, pp. 516—531.
17. **Dean E. B.** Quality Function Deployment: From the perspective of competitive advantage, 1998, available at: http://akao.larc.nasa.gov/dfc/qfd.html
18. **Saati T.** *Prinjatie reshenij. Metod analiza ierarhij.* Moscow: Radio i svjaz', 1993. 278 p. (in Russian).
19. **Bahill A. T., Chapman W. L.** A Tutorial on Quality Function Deployment. *Engineering Management Journal*, 1993, vol. 5, no. 3, pp. 24—35.
20. **King B.** *Better Designs in Half the Time: Implementing Qfd Quality Function Deployment in America*. Goal QPC Inc., 1989. 315 p.
21. **Kah-hin Chai, Jun Zhang, Kay-Chuan Tan.** National University of Singapore, ATRIZ-Based Method for New Service Design. *Journal of Service Research,* 2005, vol. 8, no. 1, pp. 48—66.
22. **Avdeeva Z. K.** Sravnitel'nyj analiz vyborochnyh kognitivnyh kart po stepeni formalizacii. *Kognitivnyj analiz i upravlenie razvitiem situacij* (*CASC'2009): Tr. mezhdunar. konf.* Moscow: IPU RAN, 2009, pp. 11—22 (in Russian).
23. **Vasantha Kandasamy W. B.** *Fuzzy Cognitive Maps and Neutrosophic Cognitive Maps.* Xiquan, 2003. 211 p.
24. **Abramova N. A.** Ekspertnaya verifikatsiya pri ispolzovanii formalnykh kognitivnykh kart. Podkhody i praktika. *Upravlenie bolshimi sistemami*, 2010, vol. 30.1, pp. 371—410 (in Russian).
25. **Axelrod R.** *The Structure of Decision: Cognitive Maps of Political Elites.* Princeton University Press, 1976. 404 p.
26. **Roberts F. S.** *Diskretnye matematicheskie modeli s prilozhenijami k social'nym, biologicheskim i jekologicheskim zadacham*. Per. s angl. Moscow: Nauka, 1986. 496 p. (in Russian).
27. **Kuznecov O. P., Kulinich A. A., Markovskij A. V.** Analiz vlijanij slabostrukturirovannymi situacijami na osnove kognitivnyj kart. *Chelovecheskij faktor v upravlenii.* Moscow: KomKniga, 2006, pp. 313—345 (in Russian).
28. **Basaker R., Saati T.** *Konechnye grafy i seti*. Moscow: Nayka, 1973. 368 p. (in Russian).
29. **Ore O.** *Teoriya grafov.* Moscow: Nauka, 1968. 352 p. (in Russian).
30. **Barsegjan A. A., Kuprijanov M. S., Stepanenko V. V., Holod N. I.** *Metody i modeli analiza dannyh: OLAP i Data Mining.* Saint Petersburg: BHV—Peterburg, 2004. 336 p. (in Russian).
31. **Pelaez E., Bowles J. W.** Using Fuzzy Cognitive Maps as a System Model for Failure Modes and Effects Analysis. *Information Sciencies*, 1996, vol. 88, pp. 177—199.
32. **Mazur G. H.** QFD for service Industries from Voice of Customer to Task Development. *The Fifth Symposium on Quality Function Deployment Novi*, Michigan June 1993, Japan Business Consultant, Ltd, 1993, pp. 485—503.
33. **Zhang W. R., Chen S. S., Chen K. H. Zhang M., Bezdek J. C.** On NPN Logic, *Proc. 18th IEEE Internat. Symp.* MVL, Palma de Malorca, Spain, 1988, pp. 381—388.
34. **Lin Z., Zhang J. Y.** Interrogating the structure of fuzzy cognitive maps. *Soft Computing*, 2003, vol. 7, pp. 148—153.

УДК 004.422.63 + 519.17

М. А. Крышень, ст. препод., e-mail: kryshen@cs.petrsu.ru,
Петрозаводский государственный университет

Абстрактные типы и неизменяемые структуры данных для интерактивной визуализации графов

Представлены абстрактные типы данных и реализующие их эффективные неизменяемые структуры данных, обеспечивающие построение гибкой интерактивной системы визуализации графов в функциональной парадигме программирования. Разрабатываемая система используется для решения задачи визуализации информационно-коммуникационной инфраструктуры организации (граф вычислительной сети вместе с пространственной и организационной структурами). Вершины (элементы информационно-коммуникационной инфраструктуры) являются сложными интерактивными объектами, взаимодействие с которыми может приводить к выполнению произвольных функций, включая изменение графа и параметров визуализации.

Ключевые слова: *абстрактные типы данных, структуры данных, неизменяемость, функциональное программирование, Clojure, визуализация, графы, интерактивность, программное обеспечение*

Введение

Во многих областях научно-технической деятельности, где применяется визуализация графов, дополнительное преимущество дает возможность работы с визуальным представлением в интерактивном режиме, допускающем изменение параметров визуализации или самого графа и быстрое получение обновленного представления.

Коллективом с участием автора разрабатывается экспериментальная программная платформа для исследования моделей и методов сетевого управления [1]. Одной из задач этого проекта является визуализация информационно-коммуникационной инфраструктуры (ИКТ-инфраструктуры) организации, представленной графом вычислительной сети вместе с пространственной и организационной структурами. Для решения этой задачи автором разрабатывается система интерактивной визуализации графов [2]. Одной из особенностей этой системы является получение визуального представления графа, в котором вершины (элементы ИКТ-инфраструктуры) являются сложными интерактивными объектами, взаимодействие с которыми может приводить к выполнению произвольных процедур, включая изменение графа и параметров визуализации.

В основе архитектуры системы визуализации лежат абстрактные типы данных (АТД), представляющие граф, раскладку графа и интерактивный визуальный объект. Эти АТД реализованы неизменяемыми структурами данных: вместо изменения данных, операции над структурой возвращают новое состояние — новую версию структуры данных, которая может разделять память с исходной. При этом исходная версия остается неизменной. Такие структуры данных называются устойчивыми (*persistent*) и обеспечивают возможность эффективной реализации алгоритмов в функциональной парадигме программирования.

Абстрактный тип данных задает класс абстрактных объектов, который полностью определен множеством применимых к этим объектам операций [3]. Использование такой абстракции позволяет поддерживать различные взаимозаменяемые представления графа и различные алгоритмы раскладки. На идее АТД основывается объектно-ориентированное программирование: класс можно рассматривать как АТД, совмещенный с его (возможно неполной) реализацией [4]. Соответственно, в объектно-ориентированных системах визуализации графов, таких как Gephi [5] и Tulip [6], определены АТД графа и раскладки. Например, в Gephi[1] определены АТД Graph, Node и Edge, которые представляют соответственно граф, вершину и ребро графа. Тип Graph включает операции для перечисления, добавления и удаления вершин и ребер. Типы Node и Edge позволяют связывать с вершинами и ребрами дополнительную информацию (метки, данные раскладки и др.), которая представляется объектами типа NodeData и EdgeData. Тип Edge дополнительно содержит операции для получения начала, конца, направленности и веса ребра. Типы NodeData и EdgeData являются подтипами типа Renderable, включающего операции

[1] Автор рассматривает текущую на время написания данной статьи стабильную версию Gephi 0.8.2.

для получения и установки параметров отрисовки: координат, цвета и размеров элемента. Алгоритм раскладки представлен типом Layout, который содержит операции для управления параметрами алгоритма и для выполнения его итераций. При выполнении очередной итерации алгоритм обновляет координаты элементов графа.

Предлагаемые автором АТД являются более простыми и менее зависимыми друг от друга. Абстрактный тип данных графа предполагает произвольный тип вершин, в частности, вершинами графа могут быть интерактивные визуальные объекты, представленные соответствующими абстрактными типами. Определение АТД графа не зависит от АТД раскладки, а раскладка может быть адаптирована для работы с вершинами произвольного типа. Архитектура системы визуализации, основанная на предложенных АТД, обеспечивает гибкость и независимость компонентов (представление графа, раскладка графа, отрисовка и интерактивное взаимодействие), возможность повторного использования кода.

Объектно-ориентированные реализации содержат большое количество изменяемого состояния, что затрудняет использование кода в многопоточном приложении. Представленная система визуализации реализована в функциональной парадигме программирования [7] с использованием неизменяемых структур данных, предоставляемых языком программирования Clojure [8]. Такой подход упрощает обеспечение потокобезопасности реализации и распараллеливание вычислений.

Обозначения

Будем использовать параметризованные (обобщенные) абстрактные типы, описание которых может содержать другие АТД в качестве параметров. Параметры указывают в квадратных скобках после имени типа. Для определения операций над типом будем использовать следующую запись:

имя операции: тип1 × тип2 × ... × типn → тип возвращаемого значения

Здесь тип1, ..., тип n — типы аргументов. Типом первого аргумента является АТД, которому принадлежит операция. Отдельно рассматриваются конструкторы, создающие новый экземпляр структуры данных, которая реализует АТД. Для конструктора типы аргументов могут быть любыми, но возвращаемое значение соответствует АТД создаваемой структуры данных. В рассматриваемом случае операции АТД являются функциями без побочных эффектов, которые возвращают новое состояние вместо изменения объекта.

Символом <: будем обозначать отношение тип — подтип, определенное принципом подстановки Барбары Лисков [9]. Согласно принципу подстановки замена объектов некоторого типа T объектами типа $S <: T$ не приведет к нарушению корректности программы.

Представление графа

Введем следующие параметризованные АТД:

- Graph[V, A] — граф, V — тип вершин, A — тип данных, ассоциированных с вершинами;
- Set[E] — конечное множество элементов типа E, тип данных позволяет перечислить элементы множества и проверить принадлежность элемента множеству, для значений типа Set[E] определены теоретико-множественные операции; для обозначения пустого множества типа Set[E] для произвольного E будем использовать символ ∅.

Следующие функции предоставляют доступ к элементам графа:

- vertices: Graph[V, A] → Set[V] — множество вершин графа;
- incoming: Graph[V, A] × V → Set[V] — множество вершин, являющихся началами дуг, для которых заданная вершина является концом;
- outgoing: Graph[V, A] × V → Set[V] — множество вершин, являющихся концами дуг, для которых заданная вершина является началом.

С вершинами могут быть ассоциированы данные. Для получения данных вершины используется функция data: Graph[V, A] × V → A.

Дополнительно отметим, что вершины графа могут быть *внешними* или *внутренними*[2]. Внешние вершины не могут быть изолированными и с ними не могут быть ассоциированы данные (функция data применима только ко внутренним вершинам). Функция intrinsic?: Graph[V, A] × V → {0, 1} возвращает 1 (истина), если вершина является внутренней, 0 (ложь) — если вершина является внешней.

Построение графа

Следующие функции дополняют АТД возможностью порождать новый измененный граф на основе существующего.

- into: Graph[V, A] × Graph[V, A] → Graph[V, A], into(g_1, g_2): возвращает граф, полученный добавлением к графу g_1 всех вершин и дуг графа g_2. Для каждой вершины v, принадлежащей обоим графам g_1 и g_2:

— вершина графа g_1 делается внутренней только если она была внутренней вершиной g_1 или g_2;

— данные, ассоциированные с вершиной в графе g_1, заменяются на данные из g_2 только если intrinsic?(g_2, v) = 1.

- remove: Graph [V, A] × Graph [V, A] → Graph [V, A], remove(g_1, g_2): из g_1 удаляются все дуги g_2; каждая вершина {v ∈ vertices(g_1) ∩ vertices(g_2): intrinsic?(g_2, v) = 1} становится внешней в графе g_1. Из результирующего графа исключаются все изолированные внешние вершины.

Возможность описания построения графа двумя простыми функциями достигнута за счет разделения вершин графа на внешние и внутренние. Функция into объединяет множества внутренних вершин и

[2] Предназначение такого разделения будет показано далее.

множества дуг двух графов, а функция remove вычисляет разности этих множеств. Таким образом, внешние вершины — это вершины, которые были удалены из множества внутренних вершин графа, но были оставлены во множестве всех вершин графа, так как в графе еще есть инцидентные им дуги.

В отличие от традиционных функций, изменяющих граф поэлементно, функции into и remove могут быть эффективно реализованы для неизменяемой структуры данных за счет использования временных изменяемых представлений для накапливания изменений. При этом побочные эффекты изолированы внутри этих функций и не видны пользователям структуры данных.

Базовым элементом для построения графов является граф типа звезда. Функция star: $V \times \text{Set}[V] \times \times \text{Set}[V] \times A \to \text{Graph}[V, A]$ возвращает граф типа звезда с заданным корнем, листьями, инцидентными ребрам, направленным к корню, листьями, инцидентными ребрам, направленным от корня, и данными, ассоциированными с корнем. Только корень является внутренней вершиной. Функция star^-: $V \times \text{Set}[V] \times \text{Set}[V] \to \text{Graph}[V, A]$ также возвращает граф типа звезда, однако все вершины этого графа — внешние. Для получения пустого графа используется функция empty-graph[V, A]: $\to$ Graph[V, A].

Определенные выше функции позволяют выполнять произвольные изменения графа, в том числе:

- добавление новой вершины v с данными d в граф g: into(g, star(v, ∅, ∅, d));
- удаление вершины v вместе с инцидентными ребрами из графа g: remove(g, star(v, incoming(g, v), outgoing(g, v), d)), где d — произвольное значение типа данных, ассоциированных с вершинами графа g;
- добавление ребра (u, v) к графу g: into(g, star^-(u, ∅, {v}));
- удаление ребра (u, v) из графа g: remove(g, star^-(u, ∅, {v})).

Накопление изменений

Используемая схема построения графа позволяет накапливать изменения и хранить информацию о различии графов.

Рассмотрим структуру данных GraphChange[V, A] (g_i, g_r), g_i, g_r: Graph[V, A].

Определим следующие функции:

- graph-change: Graph[V, A] × Graph[V, A] → → GraphChange[V, A] — возвращает структуру GraphChange с заданными значениями полей g_i и g_r;
- g_i, g_r: GraphChange[V, A] → Graph[V, A] — возвращают значения полей g_i и g_r соответственно;
- into_c: GraphChange[V, A] × Graph[V, A] → → GraphChange[V, A], $\text{into}_c(c, g) \equiv$ graph-change(into($g_i(c)$, g), remove($g_r(c)$, g));
- remove_c: GraphChange[V, A] × Graph[V, A] → → GraphChange[V, A], $\text{remove}_c(c, g) \equiv$ graph-change (remove($g_i(c)$, g), into($g_r(c)$, g)).

Теперь для накопления изменений вместо применения функций into и remove к целевому графу можно применять into_c и remove_c к структуре GraphChange, начиная с graph-change (empty-graph, empty-graph)[3].

Функция, применяющая накопленные изменения к целевому графу:

$$\text{apply: Graph}[V, A] \times \text{GraphChange}[V, A] \to \text{Graph}[V, A],$$

$$\text{apply}(g, c) \equiv \text{into}(\text{remove}(g, g_r(c)), g_i(c)).$$

Оптимизированный вариант АТД графа

В целях обеспечения лучшей производительности в рассматриваемой системе визуализации [2] используется вариант АТД графа DescriptorGraph[V, A, D], функции которого оперируют не вершинами, а дескрипторами (D — тип дескриптора) — временными идентификаторами вершин, которые действительны только для текущего состояния структуры данных. Дескриптор может представлять собой прямую ссылку на информацию о вершине в структуре данных, обеспечивая более быстрый доступ.

Представленные выше функции АТД Graph[V, A] реализованы на основе функций оптимизированного АТД графа. Таким образом, DescriptorGraph[V, A, D] является подтипом Graph[V, A] и представлен следующими дополнительными функциями:

- descriptors: DescriptorGraph[V, A, D] → Set[D] — множество всех дескрипторов (D — тип дескрипторов);
- incoming_d: DescriptorGraph[V, A, D] × D → Set[D];
- outgoing_d: DescriptorGraph[V, A, D] × D → Set[D];
- data_d:DescriptorGraph[V, A, D] × D → A;
- intrinsic_d: DescriptorGraph[V, A, D] × D → {0, 1}.

Связь дескрипторов с вершинами:

- descriptor: DescriptorGraph[V, A, D] × V → D;
- vertex: DescriptorGraph[V, A, D] × D → V.

Реализация АТД графа

Неизменяемая структура данных, реализующая АТД DescriptorGraph, основана на следующих неизменяемых устойчивых (*persistent*) структурах данных, предоставляемых языком программирования Clojure:

- *вектор* содержит n элементов, которым соответствуют индексы от 0 до $n - 1$, и позволяет быстро[4] получить i-й элемент;
- *ассоциативный массив* и *множество* на основе хэш-деревьев позволяют проверять принадлежность элемента-ключа структуре данных и находить значение по ключу (для ассоциативных массивов).

На рис. 1 показан пример графа. В этом графе вершины v_1, v_2 и v_3 являются внутренними и им соответствуют данные a_1, a_2 и a_3, вершина v_4 — внешняя.

---

[3] В реализации используются полиморфные функции into и remove, определенные и для Graph, и для GraphChange.

[4] Вектор в Clojure представляется деревом, каждый узел которого хранит 32 элемента. Сложность поиска элемента по индексу — $O(\log n)$. Однако, поскольку для практически используемых размеров вектора такое дерево имеет небольшую высоту, этим часто пренебрегают, предполагая, что время поиска не зависит от числа элементов.

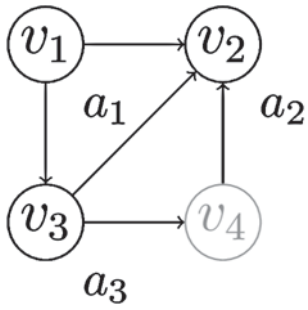


Рис. 1. Пример графа

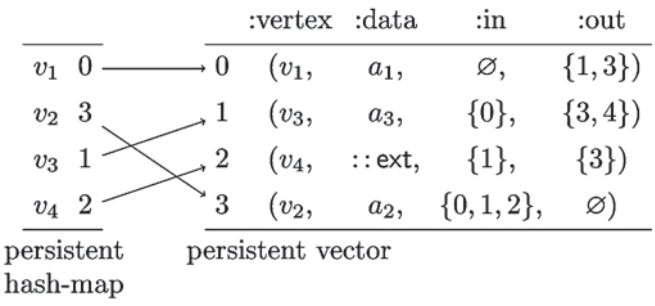


Рис. 2. Структура данных VectorGraph

Представление этого графа в структуре данных VectorGraph показано на рис. 2. Для обозначения внешней вершины здесь используется специальное значение `::ext`[5] в поле данных.

Структура VectorGraph представляет собой пару (index, contexts), где index — ассоциативный массив с ключами — вершинами и значениями — неотрицательными целыми числами. Значения index задают индексы записей в векторе contexts. Каждая запись содержит вершину, данные вершины или значение `::ext`, множества индексов, соответствующих входящим и исходящим ребрам. Записи в векторе contexts используются в качестве дескрипторов вершин при реализации DescriptorGraph. Таким образом, можно выполнять обход графа, обращаясь только к вектору contexts, избегая накладных расходов, связанных с использованием ассоциативного массива.

Для добавления вершины нужно добавить запись в contexts и соответствие в index. Для удаления вершины, соответствующей последней записи в contexts, достаточно удалить эту запись и соответствие в index. Если запись удаляемой вершины не последняя, то при удалении она замещается последней и обновляется индекс перемещенной записи в полях `:in` и `:out` записей смежных вершин.

[5] Здесь используется синтаксис языка Clojure для ключевых слов. Ключевые слова начинаются с символа ":", двойное "::" обозначает ключевое слово в текущем пространстве имен (таким образом исключается совпадение специального значения с данными пользователя).

Представление раскладки графа

Раскладка графа определяет расположение элементов графа при визуализации и, поскольку вершины графа в представленной системе могут являться сложными интерактивными объектами, должна учитывать размеры вершин. Раскладка графа Layout[*V*, *A*] <: <: Graph[*V*, *A*] является подтипом графа, предоставляющим следующие функции для получения координат и размеров вершин:

- location: Layout[*V*, *A*] × *V* → Point;
- bounds: Layout[*V*, *A*] × *V* → Interval.

Здесь Point — тип данных, представляющий точки пространства; Interval — прямоугольник со сторонами, параллельными осям координат. Дополнительно Layout[*V*, *A*] предоставляет функции для получения размеров изображения графа и поиска вершин, изображения которых пересекают заданный прямоугольник:

- layout-bounds: Layout[*V*, *A*] → Interval;
- visible-vertices: Layout[*V*, *A*] × Interval → Set[*V*].

Подтип IterativeLayout[*V*, *A*] <: Layout[*V*, *A*] представляет итеративный алгоритм раскладки графа, работающий путем постепенного уточнения координат вершин:

- update: IterativeLayout[*V*, *A*] → IterativeLayout [*V*, *A*] — результат очередной итерации;
- complete?: IterativeLayout[*V*, *A*] → {0, 1} — является ли итерация последней.

Алгоритмы раскладки могут учитывать для каждой вершины заданные ограничения на координаты, размеры и точки присоединения ребер. Функции, возвращающие эти ограничения для вершин типа *V*, составляют АТД Constraints[*V*].

Раскладка графа может являться динамической. В этом случае применение к раскладке функций into и remove порождает обновленную с учетом изменений в графе версию раскладки. Если второй аргумент в into является раскладкой, а не просто графом, динамическая раскладка может унаследовать от нее координаты новых вершин. Таким образом, можно комбинировать алгоритмы: для получения исходных координат новых вершин может использоваться быстрый алгоритм раскладки, результат работы которого будет далее уточняться итеративным алгоритмом. Например, пусть *l* — динамическая итеративная раскладка, initial-layout: Graph[*V*, *A*] → Layout[*V*, *A*] — функция, выполняющая некоторый быстрый алгоритм раскладки. Тогда обновленная добавлением нового подграфа *g* раскладка *l′* может быть получена следующим образом: *l′*= into(*l*, initial-layout(*g*)).

При использовании алгоритма раскладки графа на основе физических аналогий [10], раскладка описывается структурой данных ForceBasedLayout. Эта структура представляет собой набор (graph, tree, model, constraints), где graph — расширенная версия структуры с дополнительными полями в записях вершин для представления координат и размеров; tree — дерево поиска для вершин графа; model — энергетическая модель; constraints — реализация АТД Constraints, задает ограничения на координаты, раз-

меры вершин и точки присоединения ребер. Дерево поиска представляет иерархическое разбиение пространства координат вершин графа и используется при вычислении раскладки и для быстрого поиска подграфа, попадающего в видимую область экрана. Энергетическая модель описывает параметры раскладки и предоставляет функции для вычисления сил, действующих между вершинами. Функция update: IterativeLayout[*V*, *A*] → IterativeLayout[*V*, *A*] для структуры ForceBasedLayout возвращает новую версию структуры с обновленными координатами вершин, состоянием энергетической модели и деревом поиска.

Интерактивные визуальные объекты

Интерактивные визуальные объекты, которые могут представлять вершины графа в системе визуализации, используются аналогично элементам управления в библиотеках графического интерфейса пользователя. Основной особенностью предлагаемого подхода является то, что визуальные объекты не хранят свои координаты, размеры и принадлежность объектам-компоновщикам. Необходимые данные вычисляются при отрисовке, фиксируются и используются диспетчером событий ввода до завершения отрисовки следующего кадра.

Интерактивный визуальный объект представлен абстрактным типом данных View, который определен следующими двумя операциями:

- render: ViewContext → ViewContext — отрисовка;
- geometry: ViewContext → Geometry — получение размеров.

Тип Geometry определяет размеры и точки привязки визуального объекта. Тип ViewContext предоставляет доступ к операциям нижележащей графической библиотеки Java2D и к диспетчеру событий. Обработчики для событий ввода регистрируются во время отрисовки и действуют до завершения отрисовки следующего кадра. В реализации объекты ViewContext изменяемы и операция render может создавать побочные эффекты. Однако каждый поток (в случае многопоточной отрисовки) работает при этом с отдельным экземпляром, что позволяет исключить возникновение состояния гонки.

Для АТД View предоставляется базовый набор реализаций, которые можно комбинировать в более сложные визуальные объекты. Предоставляемые библиотекой реализации View включают текстовую подпись, растровое изображение, различные преобразования, дополняющие или изменяющие заданный визуальный объект (рамка, тень, фон, аффинные преобразования, обработка событий ввода), и компоновщики, позволяющие определенным образом расположить несколько визуальных объектов относительно друг друга. Также доступна реализация View, выполняющая асинхронную отрисовку заданного визуального объекта в другом потоке, и интерактивная область просмотра, позволяющая перемещать и масштабировать содержимое.

Поддерживаются описанные далее два способа построения сложных визуальных объектов.

- Императивное построение интерфейса выполняется подобно методу IMGUI [11] непосредственно во время отрисовки без предварительного создания иерархии компоновщиков и элементов управления. В отличие от IMGUI в рассматриваемой системе обработка событий ввода выполняется асинхронно, а не в цикле отрисовки. Например, в реализации render программа может вызвать процедуру, рисующую прямоугольник, и задать функцию, которая будет выполнена, когда пользователь поместит курсор мыши в соответствующую область. Заданное таким образом правило обработки событий будет действовать до завершения следующего цикла отрисовки, в котором отрисовка прямоугольника и определение обработчика событий могут повториться (в этом случае внешний вид и поведение соответствующей части интерфейса пользователя не изменяется).
- Декларативное описание реализуется в стиле функциональной геометрии [12]: визуальный объект определяется формулой, включающей базовые объекты и преобразования над ними.

Пример декларативного построения интерфейса: горизонтальное размещение (`hbox`) двух надписей (`label`). Первая надпись повернута на 90° (`rotate`), вторая помещена в рамку (`border`) и снабжена обработчиком событий ввода (`add-handlers`): при нажатии на кнопку мыши в стандартный поток вывода печатается "OK".

```
(hbox (rotate 90 (label "Rotated"))
      (add-handlers (border (label ("Click me")))
                    [:mouse-clicked _
                    (println "OK")]))
```

Интерфейс может быть изменяемым: функция `ref-view` возвращает визуальный объект, зависящий от значения изменяемой ссылки. В следующем примере отображается число, которое увеличивается на единицу при каждом нажатии на метку "Увеличить".

```
(let [n (atom 0)]
    (vbox (ref-view n label)
          (add-handlers (label "Увеличить")
                        [:mouse-clicked _
                        (swap! n inc)])))
```

Декларативное описание интерфейса может включать визуальные объекты или преобразования, заданные императивно. Для императивного описания визуального объекта задается реализация АТД View путем явного определения операций render и geometry.

За счет отсутствия необходимого состояния представленные интерактивные визуальные объекты об-

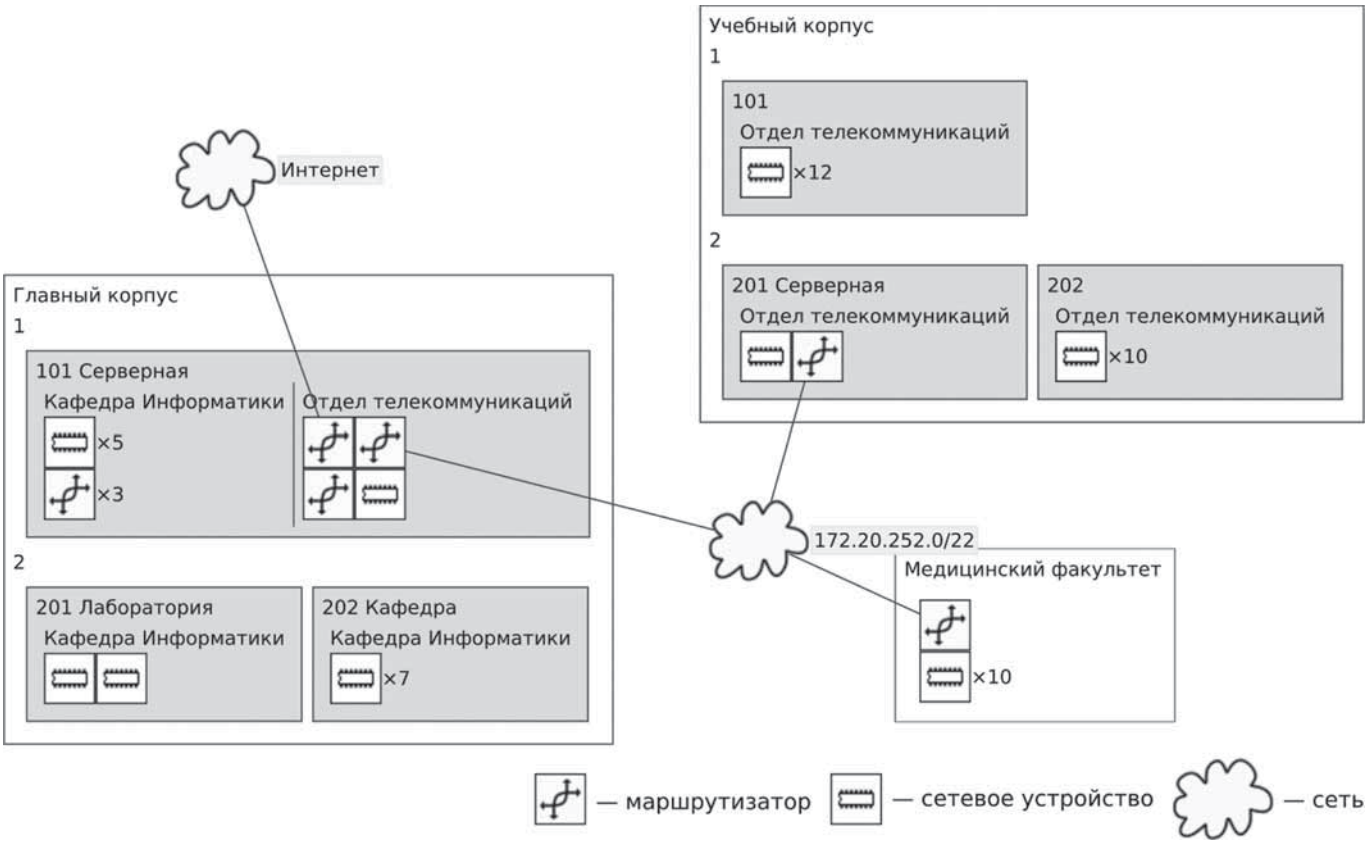


Рис. 3. Пример визуализации ИКТ-инфраструктуры

ладают следующими преимуществами перед традиционными элементами графических интерфейсов пользователя:

- меньшее использование памяти (визуализация может включать большее число объектов);
- возможность многократного использования: один объект может отображаться на экране несколько раз одновременно и динамически менять внешний вид и поведение в зависимости от контекста (например, менять уровень детализации в зависимости от масштаба);
- возможность построения графического интерфейса в непосредственном императивном (IMGUI) и в функциональном стилях;
- отсутствие необходимости синхронизировать состояния модели и представления, так как вся структура сцены визуальных объектов может определяться на основе модели динамически, во время отрисовки.

При визуализации большого графа, рядом с изображением части графа в нормальном масштабе отображается уменьшенное изображение всего графа. При этом такое изображение создается асинхронно, отдельным потоком, работающим с меньшим приоритетом. Многопоточная работа упрощается за счет отсутствия дерева компоновщиков и элементов управления, так как доступ к этому дереву требовалось бы синхронизировать, а также за счет неизменяемости основных структур данных.

Визуализация ИКТ-инфраструктуры

Для визуализации ИКТ-инфраструктуры исходный граф вычислительной сети вместе с пространственной и организационной структурами отображается в граф интерактивных визуальных объектов. Отображение определяется заданным пользователем набором правил. Правила позволяют выполнять фильтрацию и агрегацию элементов, определяют визуальное представление и интерактивное поведение вершин. Система правил описана в работе [2].

На рис. 3 показан пример визуализации части ИКТ-инфраструктуры университета[6]. Здания представлены сложными агрегированными вершинами, отображающими информацию о расположенных в здании сетевых устройствах. Для двух корпусов показаны размещение

[6] Использованы тестовые данные, не соответствующие реальной ИКТ-инфраструктуре.

устройств по этажам и комнатам, а также принадлежность устройств структурным подразделениям.

Изображенное на рис. 3 визуальное представление предполагает следующие варианты интерактивного взаимодействия:

- отображение подробных сведений об устройствах при наведении курсора;
- отображение других зданий, содержащих устройства в выбранной подсети;
- скрытие вершин графа;
- возможность разобрать здание на отдельные вершины, вплоть до отображения каждого сетевого интерфейса каждого устройства (таким образом можно увидеть конфигурацию маршрутизатора — соответствие между физическими портами и логическими сетевыми интерфейсами);
- изменение масштаба изображения, планируется реализовать динамическое изменение уровня детализации в зависимости от масштаба;
- работа с данными и их представлением посредством интерактивной среды программирования REPL языка Clojure.

Интерактивное изменение графа визуальных объектов выполняется путем замены его подграфа новым подграфом, построенным на основе измененного набора правил. Состояние визуализации полностью представлено неизменяемой структурой данных — реализацией АТД Layout[View, Set[IctElement]], где IctElement — тип данных, представляющий элементы ИКТ-инфраструктуры. За счет неизменяемости возможно моментально (без обхода и копирования структуры данных) запомнить текущее состояние и вернуться к нему впоследствии. При этом совпадающие части различных версий структуры не дублируются в памяти.

Заключение

Вычисление раскладки графа сети Петрозаводского государственного университета (5377 вершин, соответствующих каждому устройству, сетевому интерфейсу и IP-подсети) с помощью алгоритма на основе физических аналогий занимает 11,5 с на ЦП Intel i5-3330 (3,0 ГГц, 4 ядра), при этом выполняется 360 итераций алгоритма по 32 мс в среднем. Таким образом, несмотря на накладные расходы, связанные с использованием неизменяемых структур данных и высокоуровневого динамического языка программирования, система визуализации показывает достаточно хорошую производительность для решения задачи интерактивной визуализации вычислительной сети организации. На практике полезны визуализации с меньшим (за счет фильтрации и группировки элементов инфраструктуры) числом вершин.

Реализация системы визуализации выполнена на основе разработанных автором абстрактных типов данных. Такой подход позволил обеспечить гибкость и расширяемость, включая возможность реализации дополнительных алгоритмов раскладки и их комбинирование, а также возможность задавать произвольные сложные интерактивные представления для вершин графа. Использование неизменяемых структур данных и контекстно-зависимых интерактивных визуальных объектов без состояния упростило многопоточную реализацию, обеспечивающую отзывчивость интерактивной визуализации.

Исходный код библиотеки интерактивных визуальных объектов опубликован в работе [13]. В дальнейшем планируется публикация кода остальных компонентов системы.

Автор благодарен своему научному руководителю Ю. А. Богоявленскому за помощь и поддержку при выполнении данной работы.

Список литературы

1. **Крышень М. А., Козлов А. С., Тидор А. С., Богоявленский Ю. А.** Проект Nest: структурное представление системы поставщика сетевых услуг // Материалы межвузовского конкурса-конференции студентов, аспирантов и молодых ученых Северо-Запада "Технологии Microsoft в теории и практике программирования". СПб.: Изд-во СПбГПУ, 2008. С. 49—51.

2. **Крышень М. А., Богоявленский Ю. А.** Интерактивная визуализация графа ИКТ-инфраструктуры организации // Научно-технические ведомости СПбГПУ. 2011. № 3 (126). С. 172—175.

3. **Liskov B., Zilles S.** Programming with abstract data types // Proceedings of the ACM SIGPLAN symposium on very high level languages. New York, NY, USA: ACM, 1974. P. 50—59.

4. **Meyer B.** Object-Oriented Software Construction. Upper Saddle River, NJ: Prentice Hall, 1997.

5. **Bastian M., Heymann S., Jacomy M.** Gephi: An open source software for exploring and manipulating networks // International AAAI Conference on Weblogs and Social Media. 2009. P. 361—362.

6. **Auber D., Mary P., Mathiaut M.** et al. Tulip: a Scalable Graph Visualization Framework // Extraction et Gestion des Connaissances (EGC) 2010 / Ed. S. B. Yahia, J.-M. Petit. Hammamet, Tunisia: RNTI, 2010. Vol. RNTI E-19. P. 623—624.

7. **Hughes J.** Why Functional Programming Matters // Research Topics in Functional Programming / Ed. D. Turner. Addison Wesley, 1990. P. 17—42.

8. **Hickey R.** The Clojure Programming Language // Proceedings of the 2008 Symposium on Dynamic Languages. New York, NY, USA: ACM, 2008. P. 1:1—1:1.

9. **Liskov B.** Keynote address — data abstraction and hierarchy // SIGPLAN Not. 1987. Vol. 23, N 5. P. 17—34.

10. **Fruchterman T. M. J., Reingold E. M.** Graph drawing by force-directed placement // Softw. Pract. Exper. 1991. Vol. 21, N 11. P. 1129—1164.

11. **Barrett S.** Immediate Mode GUIs // Game Developer. 2005. September. P. 34—36.

12. **Henderson P.** Functional geometry // Higher Order and Symbolic Computation. 2002. Vol. 15. P. 349—365.

13. **Kryshen M.** Indyvon. [Electronic resource]. URL: https://bitbucket.org/kryshen/indyvon (дата обращения: 10.03.2015).

M. A. Kryshen, Senior Lecturer, e-mail: kryshen@cs.petrsu.ru, Petrozavodsk State University

Abstract Data Types and Immutable Data Structures for Interactive Graph Visualization

In various applications of graph visualization, including network management, additional benefits arise from the ability to interactively manipulate the graph and the parameters of its visual representation. In the context of the task of IT infrastructure visualization, an interactive graph visualization system is being developed. The system uses a user-definable set of rules to translate source data describing the IT infrastructure into a graph of complex interactive vertices. By manipulating the vertices, the user can trigger changes in the graph structure or parameters of the visualization.

The architecture of the visualization system is based on abstract data types for graph, graph layout, and interactive visual objects that represent the graph vertices. These abstract data types are designed to ensure composability of system components and allow efficient implementation with immutable data structures. Graphs are constructed by combining graphs instead of using operations on individual graph elements. Graph layout algorithms are composable: a fast algorithm implementation can provide predictable initial locations for new vertices, then the layout can be refined by an iterative algorithm. Interactive visual objects are composable, stateless, and context-dependant. The use of immutable data structures allowed to reduce and localize shared state that needs to be synchronized, which simplifies multi-threaded implementation needed to achieve good responsiveness of the interactive system.

Keywords: *abstract data types, immutable data structures, functional programming, Clojure, graph visualization, interactive visualization, software system*

References

1. **Kryshen M. A., Kolosov A. S., Tidor A. S., Bogoyavlenskiy Ju. A.** Proekt Nest: strukturnoe predstavlenie sistemy postavshika setevyh uslug. *Materialy mezhvuzovskogo konkursa-konferencii studentov, aspirantov i molodyh uchenyh Severo-Zapada "Tehnologii Microsoft v teorii i praktike programmirovanija"*. Saint Petersburg: Izdatel'stvo SPbGPU, 2008, pp. 49—51 (in Russian).

2. **Kryshen M. A., Bogoyavlenskiy Ju. A.** Interaktivnaja vizualizacija grafa IKT-infrastruktury organizacii. *Nauchno-tehnicheskie vedomosti SPbGPU*, 2011, no 3 (126), pp. 172—175 (in Russian).

3. **Liskov B., Zilles S.** Programming with abstract data types. *Proceedings of the ACM SIGPLAN symposium on very high level languages*. New York, NY, USA: ACM, 1974, pp. 50—59.

4. **Meyer B.** *Object-Oriented Software Construction*. Upper Saddle River, NJ: Prentice Hall, 1997.

5. **Bastian M., Heymann S., Jacomy M.** Gephi: An open source software for exploring and manipulating networks. *International AAAI Conference on Weblogs and Social Media*. 2009, pp. 361—362.

6. **Auber D., Mary P., Mathiaut M. et al.** Tulip: a Scalable Graph Visualization Framework. *Extraction et Gestion des Connaissances* (EGC) 2010 / Ed. S. B. Yahia, J.-M. Petit Hammamet, Tunisia: RNTI, 2010, vol. RNTI E-19, pp. 623—624.

7. **Hughes J.** Why Functional Programming Matters. *Research Topics in Functional Programming* / Ed. D. Turner. Addison Wesley, 1990. P. 17.

8. **Hickey R.** The Clojure Programming Language. *Proceedings of the 2008 Symposium on Dynamic Languages*. New York, NY, USA: ACM, 2008, pp. 1:1—1:1.

9. **Liskov B.** Keynote address — data abstraction and hierarchy. *SIGPLAN Not.*, 1987, vol. 23, no 5, pp. 17—34.

10. **Fruchterman T. M. J., Reingold E. M.** Graph drawing by force-directed placement. *Softw. Pract. Exper.,* 1991, vol. 21, no 11, pp. 1129—1164.

11. **Barrett S.** Immediate Mode GUIs. *Game Developer*, 2005, September, pp. 34—36.

12. **Henderson P.** Functional geometry. *Higher Order and Symbolic Computation,* 2002, vol. 15, pp. 349—365.

13. **Kryshen M.** Indyvon. [Electronic resource], available at: https://bitbucket.org/kryshen/indyvon (data of access 10.03.2015).

ИНФОРМАЦИЯ

УДК 629.73.02; 629.73.05/.06; 535.643

И. О. Жаринов, д-р техн. наук, доц., зав. каф., Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики (НИУ ИТМО), руководитель учебно-научного центра, Санкт-Петербургское ОКБ "Электроавтоматика" им. П. А. Ефимова, e-mail: igor_rabota@pisem.net,
О. О. Жаринов, канд. техн. наук, доц., Санкт-Петербургский государственный университет аэрокосмического приборостроения (ГУАП)

Исследование влияния внешней освещенности на колориметрические характеристики воспринимаемого наблюдателем изображения в авионике

Рассмотрена задача и в рамках ее решения разработан способ программной коррекции компонентов кодов цветов, входящих в состав цветовой палитры бортового средства индикации. Показано, что цветовая палитра, полученная при каком-либо одном источнике искусственной освещенности в составе автоматизированного рабочего места, не может быть использована при всех условиях эксплуатации. В таких условиях цветовая температура источника естественной освещенности меняется в широких пределах и отличается от цветовой температуры источника искусственной освещенности. Для индикации изображения в условиях воздействия на экран различных источников света предложен метод и расчетные формулы коррекции кодов цветовой палитры. Показано, что соотношение яркости цвета изображения на жидкокристаллической панели и яркости цвета, образованного диффузным отражением света источника внешней освещенности, является ключевым не только для расчета контраста индикации на экране заданного элемента изображения, но и для цветового охвата, воспринимаемого наблюдателем в целом.

Ключевые слова: *индикация, авионика, координаты цветности, внешняя освещенность, коррекция*

Введение

Практический опыт эксплуатации бортовых индикаторов класса МФЦИ (многофункциональные цветные индикаторы), выполненных на базе плоских жидкокристаллических (ЖК) панелей [1], показал, что надежность восприятия наблюдателем пилотажно-навигационной информации в значительной степени зависит от светотехнических характеристик МФЦИ и условий наблюдения.

Светотехнические характеристики МФЦИ определяются яркостью изображения на экране в цветах, заданных в цветовой палитре программного обеспечения, а также коэффициентами диффузного и зеркального отражения. Условия наблюдения предполагают восприятие информации наблюдателем на фоне внешнего источника света (естественного или искусственного), создающего в плоскости экрана ЖК панели повышенный уровень освещенности. Наличие внешней освещенности оказывает заметное влияние на восприятие индицируемой информации и в ряде случаев приводит к инверсии отображаемого цвета, что недопустимо в авиационном приборостроении.

Для снижения влияния спектрального состава и уровня внешней освещенности на качество восприятия изображения используют различные подходы, основанные:

- на повышении яркости изображения МФЦИ за счет использования предельных электрических режимов ламп (светодиодов) подсвета ЖК панелей [2, 3];
- на использовании специализированных светопоглощающих покрытий, снижающих значения коэффициентов диффузного и зеркального отражения [4—6];
- на подборе по результатам теоретических исследований и практических экспериментов на све-

тотехнических стендах программных компонентов кодов цветов [7—9], составляющих цветовую палитру бортового средства индикации и обладающих наилучшими характеристиками восприятия для наблюдателя.

Каждый из этих подходов в отдельности имеет ограничения по возможности своего применения. Так, существенное повышение яркости изображения на экране ЖК панели резко снижает ресурс прибора. Введение в конструкцию МФЦИ антибликовых и антиотражающих пленок значительно ухудшает отражательные способности ЖК панелей и повышает контраст изображения. Однако такие покрытия, которые наносятся с использованием клея на промышленно изготавливаемые ЖК панели, неустойчивы к плесневым грибам, повышенной влажности и соляному туману. Определение программных кодов цветов на практике осуществляется с использованием осветителей только одного типа в предположении об эквивалентности спектров излучения искусственного и естественных источников света во всех возможных условиях эксплуатации летательного аппарата.

В связи с вышеизложенным актуальной является задача разработки и исследования эффективности метода, сочетающего в себе все три перечисленных выше подхода. Такой метод должен обеспечивать восприятие пилотажно-навигационной информации наблюдателем для всех возможных видов естественных источников света, которые встречаются на практике, с учетом интенсивности их излучения.

Смещение координат цветности воспринимаемого наблюдателем изображения

Спектральный состав источника света, которым освещается ЖК панель МФЦИ в составе автоматизированного рабочего места (АРМ), в значительной степени влияет на колориметрические характеристики цвета, воспринимаемого наблюдателем [8]. Один и тот же цвет, индицируемый на экране МФЦИ, воспринимается наблюдателем по-разному в условиях наличия или отсутствия внешней освещенности и источников внешней освещенности, обладающих различной цветовой температурой T_k, измеряемой в Кельвинах. В таблице приведены колориметрические характеристики типовых источников естественной и искусственной освещенности.

Смещение (x, y)-координат цветности элементов изображения, индицируемых на ЖК панели МФЦИ, и сужение цветового охвата, воспринимаемого наблюдателем, обусловлено влиянием свойств [10] внешней отражающей поверхности ЖК панели. Часть внешней освещенности, создаваемой источником света в плоскости ЖК панели, отражается в направлении наблюдателя пропорционально ненулевому значению коэффициента $\rho_Д$ диффузного отражения. Влияние коэффициента зеркального отражения ЖК панели может быть сведено к нулю за счет конструктивного размещения МФЦИ в составе кабины летательного аппарата, при котором внешний падающий свет зеркально переотражается не в направлении наблюдателя.

Свет, диффузно отраженный от ЖК панели в направлении наблюдателя, создает дополнительный цвет, который складывается с цветом изображения, индицируемого на ЖК панели. Таким образом, наблюдатель вместо цвета изображения воспринимает в каждой точке экрана смесь двух разнояркостных цветов: цвета изображения и цвета, отраженного от экрана ЖК панели.

Координаты цвета, излучаемого ЖК панелью, определяются преобразованием Грассмана [11]:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} X_r & X_g & X_b \\ Y_r & Y_g & Y_b \\ Z_r & Z_g & Z_b \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}; \quad \begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} X_r & X_g & X_b \\ Y_r & Y_g & Y_b \\ Z_r & Z_g & Z_b \end{bmatrix}^{-1} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}, \qquad (1)$$

где X, Y, Z — координаты цвета; X_r, X_g, X_b, Y_r, Y_g, Y_b, Z_r, Z_g, Z_b — компоненты цвета, определенные Международной комиссией по освещению; R, G, B — коды (в десятичной системе счисления) компонентов основных цветов (красный, зеленый, синий) в системе RGB (R — Red, G — Green, B — Blue). Компоненты X_r, Y_r, Z_r представляют собой веса десятичных кодов красного цвета, компоненты X_g, Y_g, Z_g и X_b, Y_b, Z_b — веса кодов зеленого и синего цветов соответственно. Компоненты цвета определяют модель цветопередачи (профиль) ЖК панели.

Координаты цвета, создаваемого диффузно переотраженным светом источника внешней освещенности, определяются из следующих соотношений [12]:

$$\begin{cases} X = \int\limits_{\lambda=380}^{\lambda=760} E(\lambda)\bar{x}(\lambda)\rho_Д d\lambda \\ Y = \int\limits_{\lambda=380}^{\lambda=760} E(\lambda)\bar{y}(\lambda)\rho_Д d\lambda \\ Z = \int\limits_{\lambda=380}^{\lambda=760} E(\lambda)\bar{z}(\lambda)\rho_Д d\lambda \end{cases}, \qquad (2)$$

где $E(\lambda)$ — относительное распределение спектральной плотности освещенности источника света; $\bar{x}(\lambda)$, $\bar{y}(\lambda)$, $\bar{z}(\lambda)$ — удельные координаты сложения для основных цветов; λ — длина световой волны.

Интегрирование (2) выполняется в диапазоне видимых длин волн путем суммирования произведений подинтегральных функций:

$$\begin{cases} X = \sum\limits_{\lambda=380}^{\lambda=760} E(\lambda)\bar{x}(\lambda)\rho_Д \Delta\lambda \\ Y = \sum\limits_{\lambda=380}^{\lambda=760} E(\lambda)\bar{y}(\lambda)\rho_Д \Delta\lambda \\ Z = \sum\limits_{\lambda=380}^{\lambda=760} E(\lambda)\bar{z}(\lambda)\rho_Д \Delta\lambda \end{cases}. \qquad (3)$$

Источники естественной и искусственной освещенности

T_k, К	Координаты цветности, ед.		Максимальная освещенность, Лк	Естественный источник внешней освещенности	Эквивалентный искусственный источник внешней освещенности
2000	x	0,5267	10...20	Свет солнца (накануне восхода и заката солнца)	Натриевая лампа высокого давления
	y	0,4133			
3500	x	0,4053	3500	Свет солнца за час до заката и через час после восхода	Кварцевая галогеновая лампа
	y	0,3907			
4000	x	0,3804	1	Лунный свет	Флуоресцентная лампа
	y	0,3767			
4300	x	0,3681	10 000	Свет солнца за два часа до заката и через два часа после восхода	Ртутные лампы, металлогалогеновые лампы
	y	0,3687			
4800	x	0,3509	75 000	Прямой солнечный свет	Ксеноново-дуговая лампа
	y	0,3562			
5000	x	0,3451	25 000	Ясное полуденное небо	
	y	0,3516			
5400	x	0,3347		Свет летнего полуденного неба в пасмурный день	Лампа дневного света, электролюминесцентная лампа, ртутная лампа
	y	0,3430			
6000	x	0,3221	16 000	Свет от облачного неба	
	y	0,3317			
6500	x	0,3135		Облачное небо, рассеянный дневной свет	
	y	0,3236			
8000	x	0,2952	30 000	Свет от летнего безоблачного неба в северных широтах	Специализированные лампы, применяемые преимущественно для освещения аквариумов
	y	0,3047			
15 000	x	0,2637	30 000	Голубое небо в морозную погоду	
	y	0,2672			
20 000	x	0,2564	70 000	Синее небо в районе полярного полюса	
	y	0,2576			

Для стандартных люминесцентных ламп искусственного света в формуле (3) принимается $\Delta\lambda = 10$ нм. Для ламп, в спектре излучения которых имеется неравномерность в интервале менее 10 нм, интервал $\Delta\lambda$ выбирается на уровне 5 нм. Удельные координаты сложения $\bar{x}(\lambda)$, $\bar{y}(\lambda)$, $\bar{z}(\lambda)$, необходимые для расчета, приведены в ГОСТ 23198—94 "Лампы электрические. Методы измерения спектральных и цветовых характеристик" в виде таблицы.

Для исключения влияния на результаты расчетов яркости цвета координаты цвета XYZ могут быть нормированы по значению Y. Координаты цветности (x, y) для цвета, создаваемого диффузным переотражением внешнего светового потока, а также для индицируемого цвета, определяются по следующим правилам:

$$x = \frac{X}{X+Y+Z},\quad y = \frac{Y}{X+Y+Z},\quad z = \frac{Z}{X+Y+Z},$$
$$X = \frac{xY}{y},\quad Z = \frac{(1-x-y)Y}{y}. \qquad (4)$$

Яркость смеси двух цветов, воспринимаемых наблюдателем, суммируется. Яркость изображения $L_и$ определяется яркостью ЖК панели в выбранном цвете. Яркость цвета, диффузно отраженного от поверхности ЖК панели, определяется по ГОСТ 52870—07:

$$L_{вн} = \frac{\rho_Д E_{вн}}{\pi}, \qquad (5)$$

где $E_{вн}$ — уровень освещенности плоскости экрана, измеряемый люксметром.

Коэффициент диффузного отражения $\rho_Д$ определяется экспериментально в выключенном состоянии ЖК панели по методике [13], предусматривающей формирование в плоскости экрана заданного уровня $E_{вн}$ и измерения яркомером $L_{вн}$. Снижение коэффициента $\rho_Д$ возможно за счет нанесения на экран специализированных покрытий [4—6].

Таким образом, яркость смеси $L_с = L_{вн} + L_и$, где $L_{вн}$ определяется только коэффициентом отражения $\rho_Д$ экрана ЖК панели и уровнем внешней освещенности $E_{вн}$. Уровень $E_{вн}$ в эксплуатации определяет система автоматической регулировки яркости [14], расположенная в лицевой части МФЦИ.

Анализ технической документации ЖК панелей показывает, что в зависимости от назначения ЖК панели, яркость свечения экрана в белом цвете варьируется следующим образом: 300 кд/м^2 ⩽ $L_и$ ⩽ 300 кд/м^2, а значение коэффициента диффузного отражения $\rho_д$ находится в пределах 0,1...1,5 %. Таким образом, при уровне внешней освещенности до 100 кЛк, яркости ЖК панели в белом цвете $L_и$ = 300 кд/м^2, $\rho_Д$ = 1 % и установке такой ЖК панели в МФЦИ, соотношение собственной яркости (в белом цвете) и яркости диффузно отраженного цвета будет следующим: $L_и$:$L_{вн}$ ≈ 1:1. В то же время для $L_и$ = 1000 кд/м^2, $\rho_Д$ = 0,1 %: $L_и$:$L_{вн}$ ≈ 1:30. Соотношение яркостей в других цветах определяется яркостью ЖК панели в этих цветах при сохранении яркости (5) для заданного уровня внешней освещенности $E_{вн}$. Результаты измерения яркости различных моделей МФЦИ приведены в работе [7], результаты оценки их коэффициента $\rho_Д$ — в работе [13].

Координаты цветности интегрального цвета (смеси), воспринимаемого наблюдателем и образованного суммой двух цветов, заданных колориметрическими характеристиками (x_1, y_1, Y_1), (x_2, y_2, Y_2), определяются по следующим выражениям:

$$x = \frac{x_1\frac{Y_1}{y_1} + x_2\frac{Y_2}{y_2}}{\frac{Y_1}{y_1} + \frac{Y_2}{y_2}}, \quad y = \frac{Y_1 + Y_2}{\frac{Y_1}{y_1} + \frac{Y_2}{y_2}}, \quad Y = Y_1 + Y_2. \qquad (6)$$

Цветовой охват наблюдателя в условиях воздействия внешней освещенности

Для исследования цветового охвата наблюдателя в условиях восприятия изображения при повышенном уровне внешней освещенности была проведена серия экспериментов и написана специализированная компьютерная программа в среде Mathcad 15.0.

Эксперименты проводили на светотехнической установке (рис. 1) с использованием серийно изготовленного МФЦИ. Пример тестового индикационного кадра на ЖК панели МФЦИ приведен на рис. 2.

Светотехническая установка представляет собой испытательный стенд, предназначенный для измерения светотехнических характеристик бортового индикационного оборудования (МФЦИ, пульты управления и индикации и др.).

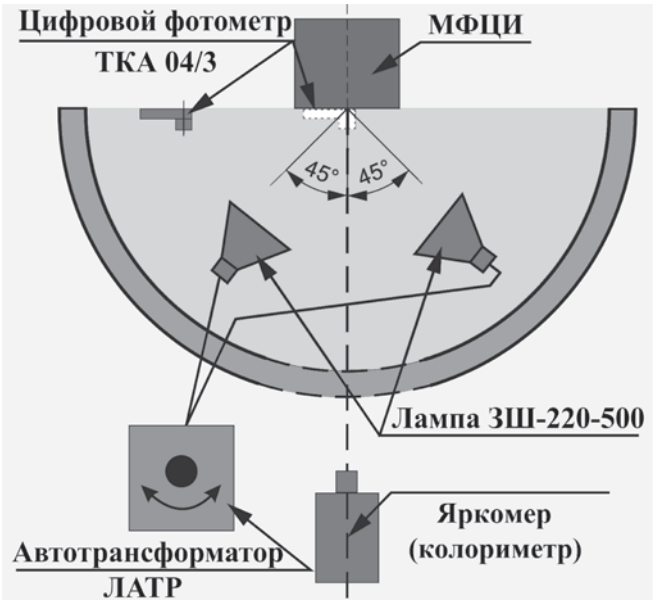


Рис. 1. Схема светотехнической установки

Светотехническая установка включает:

- цифровой фотометр, осуществляющий измерение уровня внешней освещенности в плоскости экрана МФЦИ;
- яркомер, осуществляющий измерение яркости изображения на экране;

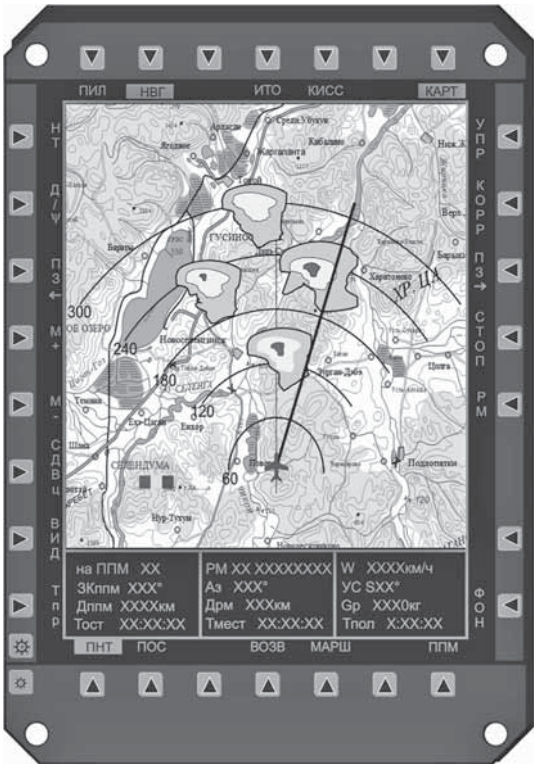


Рис. 2. Пример тестового индикационного кадра на ЖК панели МФЦИ

- колориметр, осуществляющий измерение колориметрических характеристик изображения на экране МФЦИ;
- две осветительные лампы, создающие в плоскости экрана освещенность от искусственного источника света, близкого по спектральному составу излучения к естественному источнику света;
- линейный автотрансформатор (ЛАТР), осуществляющий регулировку напряжения питания осветительных ламп для создания в плоскости экрана МФЦИ заданного уровня внешней освещенности;
- элементы конструкции, обеспечивающие крепление оборудования.

Светотехническая установка позволяет создавать регулируемый с помощью ЛАТР уровень внешней освещенности $E_{\text{вн}}$ в плоскости экрана МФЦИ 0...75 кЛк и измерять уровень яркости, координаты цвета и координаты цветности каждого элемента изображения, индицируемого на экране МФЦИ, и яркость цвета фона.

Способ крепления яркомера и колориметра позволяет перемещать измерительные приборы вверх-вниз, влево-вправо для обеспечения возможности измерения характеристик изображения в любой точке экрана МФЦИ. Осветительные лампы располагаются под углом 45° по отношению к нормали к плоскости ЖК панели МФЦИ. Измерение яркости и координат цвета (цветности) осуществляется по нормали к экрану МФЦИ.

Расстояние от осветительных ламп до МФЦИ (примерно 700 мм) исключает возможность влияния теплового воздействия от разогрева ламп на ЖК панель МФЦИ и обеспечивает создание в плоскости экрана МФЦИ рассеянного света. Основу элементов конструкции стенда составляет отражатель.

Компьютерная программа в среде Mathcad 15.0, разработанная специально для проведения исследования, обеспечивает расчет:

— по формулам (1), (4) координат цветности красного, зеленого и синего цветов, определяющих положение треугольника цветового охвата ЖК панели на *XY*-плоскости, и отображение границ треугольника цветового охвата;

— по формулам (2)—(4) координат цветности цвета, образованного диффузно отраженным внешним светом от поверхности ЖК панели;

— по формуле (6) компонента относительной яркости, расчет по формуле (6) и отображение на *XY*-плоскости координат цветности для аддитивной смеси, образованной смешением индицируемого на ЖК панели и отраженного цветов.

В качестве индицируемого цвета в программе задается (в цикле) поочередно каждый цвет, входящий в номенклатуру воспроизводимых на ЖК панели цветов. В качестве отраженного цвета используется белый цвет от источника внешней освещенности, заданный коррелированной цветовой температурой T_k.

Координаты цветности белого цвета от источника внешней освещенности определяются в программе из следующих соотношений:

$$x(T_k) = \begin{cases} \dfrac{-4,6070 \cdot 10^9}{T_k^3} + \dfrac{2,9678 \cdot 10^6}{T_k^2} + \dfrac{0,09911 \cdot 10^3}{T_k} + \\ +0,244063, \quad 2000\ \text{K} \leqslant T_k \leqslant 7000\ \text{K} \\ \dfrac{-2,0064 \cdot 10^9}{T_k^3} + \dfrac{1,9018 \cdot 10^6}{T_k^2} + \dfrac{0,24748 \cdot 10^3}{T_k} + \\ +0,237040, \quad 7000\ \text{K} < T_k \leqslant 25\,000\ \text{K} \end{cases}$$

$$y(T_k) = -3x^2(T_k) + 2,87x(T_k) - 0,275.$$

Результатом работы компьютерной программы являются семейства графиков цветовых охватов наблюдателя, воспринимающего изображения на ЖК панели МФЦИ в условиях внешней освещенности от различных источников белого света, заданных коррелированной цветовой температурой T_k.

Графики цветовых охватов приведены на рис. 3 для шести различных значений температур T_k. Расчет координат цветности вершин треугольника цветового охвата выполнен по следующим формулам:

$$R\colon \left\{x_R = X_r/(X_r + Y_r + Z_r), \quad y_R = Y_r/(X_r + Y_r + Z_r)\right\},$$

$$G\colon \left\{x_G = X_g/(X_g + Y_g + Z_g), \quad y_G = Y_g/(X_g + Y_g + Z_g)\right\},$$

$$B\colon \left\{x_B = X_b/(X_b + Y_b + Z_b), \quad y_B = Y_b/(X_b + Y_b + Z_b)\right\},$$

где (x_R, y_R), (x_G, y_G), (x_B, y_B) — координаты цветности точки соответственно красного (R), зеленого (G) и синего (B) цветов.

Анализ графиков показывает, что сужение цветового охвата, воспринимаемого наблюдателем, обусловлено смещением координат цветности изображения на МФЦИ за счет влияния компонентов отраженного цвета (x_2, y_2, Y_2). Компоненты (x_2, y_2) смещают координаты цветности (x_1, y_1) индицируемого изображения, яркостная компонента Y_2 в значительной степени способствует сужению цветового охвата наблюдателя. Смещение координат цветности цветов осуществляется в область белого цвета за счет образования в смеси цветов, входящих в цветовой охват, повышенного числа оттенков серого цвета.

Оценочные графики смещения координат цветности, полученные в результате моделирования, приведены на рис. 4, см. третью сторону обложки. Штрих-пунктирной линией на графиках показано истинное значение координаты цветности для индицируемого цвета с учетом заданного профиля модели цветопередачи ЖК панели.

Для источников повышенной внешней освещенности с "теплым" белым цветом характерно существенное смещение воспринимаемых наблюдателем цветов из области насыщенного синего и зеленого цветов в направлении точки белого цвета, определяемой источником освещенности. Таким образом, в условиях повышенной освещенности от источника "теплого" белого света ухудшается восприятие наблюдателем зеленого и синего цветов и их оттен-

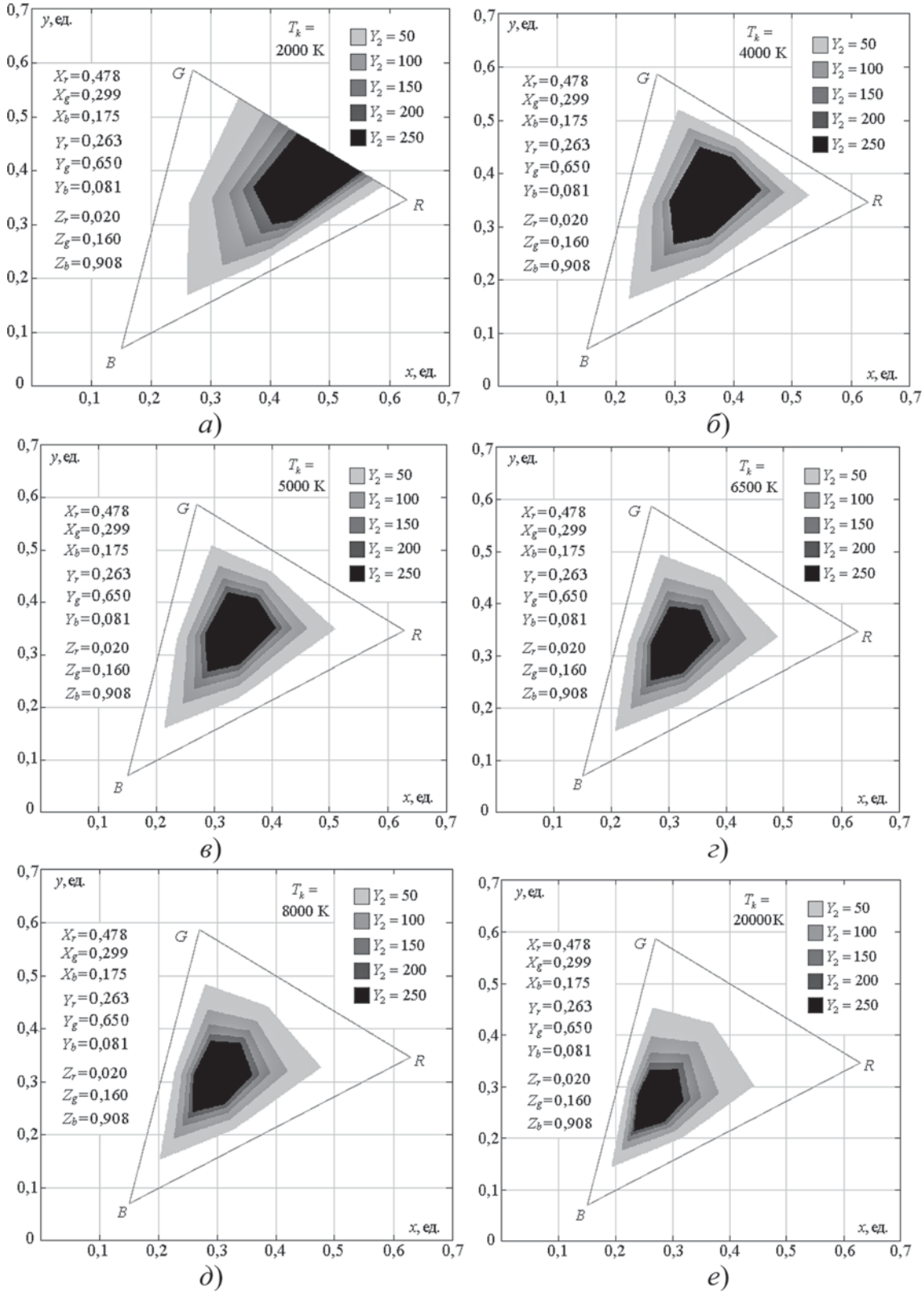


Рис. 3. Цветовой охват, воспринимаемый наблюдателем, в условиях воздействия внешней освещенности с цветовой температурой источника:

а — T_k = 2000 K; *б* — T_k = 4000 K; *в* — T_k = 5000 K; *г* — T_k = 6500 K; *д* — T_k = 8000 K; *е* — T_k = 20 000 K

ков, которые образуются незначительным добавлением в код *RGB* компонента красного цвета. Оттенки красного цвета различаются наблюдателем в полной мере при достаточном уровне яркостного контраста изображения.

Для источников повышенной внешней освещенности с "ровным" белым цветом, а также с белым цветом, координаты точки T_k которого совпадают на *XY*-плоскости с координатами точки белого цвета треугольника цветового охвата, свойственного ЖК панели МФЦИ, смещению воспринимаемых наблюдателем цветов в равной степени подвержены основные цвета (красный, зеленый, синий) и желтый, голубой, пурпурный цвета. В связи с этим обстоятельством использование в программном коде *RGB* цветовой палитры МФЦИ "чистых" цветов нежелательно.

Для источников повышенной внешней освещенности с "холодным" белым цветом характерно существенное смещение воспринимаемых наблюдателем цветов из области насыщенного красного и зеленого цветов в направлении точки белого цвета, определяемой источником освещенности. Таким образом, в условиях повышенной освещенности от источника "холодного" белого света ухудшается восприятие наблюдателем зеленого и красного цветов и их оттенков, которые образуются незначительным добавлением в код *RGB* компонента синего цвета. Оттенки синего цвета различаются наблюдателем в полной мере при достаточном уровне яркостного контраста изображения.

Важно отметить, что преобразование цветового охвата наблюдателя характерно для источников внешней освещенности с различной цветовой температурой. В связи с этим обстоятельством цветовая палитра МФЦИ должна разрабатываться с учетом видов спектров внешней освещенности, обусловленных всеми возможными условиями эксплуатации летательного аппарата.

Преобразование компонентов программных кодов цветовой палитры

Преобразование цветовой палитры МФЦИ, заданной кодами *RGB*, под условия эксплуатации летательного аппарата при освещенности ЖК панели источниками белого света различной цветовой температуры осуществляется следующим образом.

В процессе светотехнических испытаний на АРМ, в состав которого входят источники искусственной освещенности (лампы) одного типа, формируется цветовая палитра МФЦИ. Каждый цвет этой палитры смешивается с диффузно отраженным цветом, в результате чего смесь цветов (x_{opt}, y_{opt}, Y_{opt}) характеризуется повышенными визуальными характеристиками восприятия (контраст) для наблюдателя и соответствует нормативно-технической документации, действующей в авиационной промышленности. В то же время цвет смеси (x_{opt}, y_{opt}, Y_{opt}) может быть получен различными комбинациями координат цветности изображения и координат цветности диффузно отраженного цвета.

Таким образом, в соответствии с формулой (6) справедливо:

$$\frac{x_{\text{кор}}\frac{Y_{\text{кор}}}{y_{\text{кор}}}+x_{\text{ест}}\frac{Y_{\text{ест}}}{y_{\text{ест}}}}{\frac{Y_{\text{кор}}}{y_{\text{кор}}}+\frac{Y_{\text{ест}}}{y_{\text{ест}}}}=x_{opt}=\frac{x_{\text{пал}}\frac{Y_{\text{пал}}}{y_{\text{пал}}}+x_{\text{иск}}\frac{Y_{\text{иск}}}{y_{\text{иск}}}}{\frac{Y_{\text{пал}}}{y_{\text{пал}}}+\frac{Y_{\text{иск}}}{y_{\text{иск}}}}\Rightarrow$$

$$\Rightarrow x_{\text{кор}}=\frac{x_{\text{пал}}\frac{Y_{\text{пал}}}{y_{\text{пал}}}+x_{\text{иск}}\frac{Y_{\text{иск}}}{y_{\text{иск}}}-x_{\text{ест}}\frac{Y_{\text{ест}}}{y_{\text{ест}}}}{\frac{Y_{\text{пал}}}{y_{\text{пал}}}+\frac{Y_{\text{иск}}}{y_{\text{иск}}}-\frac{Y_{\text{ест}}}{y_{\text{ест}}}}, \quad (7)$$

$$\frac{Y_{\text{кор}}+Y_{\text{ест}}}{\frac{Y_{\text{кор}}}{y_{\text{кор}}}+\frac{Y_{\text{ест}}}{y_{\text{ест}}}}=y_{opt}=\frac{Y_{\text{пал}}+Y_{\text{иск}}}{\frac{Y_{\text{пал}}}{y_{\text{пал}}}+\frac{Y_{\text{иск}}}{y_{\text{иск}}}}\Rightarrow$$

$$\Rightarrow y_{\text{кор}}=\frac{Y_{\text{пал}}+Y_{\text{иск}}-Y_{\text{ест}}}{\frac{Y_{\text{пал}}}{y_{\text{пал}}}+\frac{Y_{\text{иск}}}{y_{\text{иск}}}-\frac{Y_{\text{ест}}}{y_{\text{ест}}}}, \quad (8)$$

$$Y_{\text{кор}}+Y_{\text{ест}}=Y_{\text{opt}}=Y_{\text{пал}}+Y_{\text{иск}}\Rightarrow$$

$$\Rightarrow Y_{\text{кор}}=Y_{\text{пал}}+Y_{\text{иск}}-Y_{\text{ест}}, \quad (9)$$

где ($x_{\text{пал}}$, $y_{\text{пал}}$, $Y_{\text{пал}}$) — колориметрические характеристики выводимого на экран МФЦИ цвета (красный, зеленый, синий, белый и др.) из цветовой палитры; ($x_{\text{иск}}$, $y_{\text{иск}}$, $Y_{\text{иск}}$) — колориметрические характеристики цвета, диффузно отраженного от поверхности ЖК панели МФЦИ в условиях светотехнического стенда АРМ для искусственного освещения; ($x_{\text{ест}}$, $y_{\text{ест}}$, $Y_{\text{ест}}$) — колориметрические характеристики цвета, диффузно отраженного от поверхности ЖК панели МФЦИ в условиях воздействия естественной освещенности в эксплуатации; ($x_{\text{кор}}$, $y_{\text{кор}}$, $Y_{\text{кор}}$) — колориметрические характеристики скорректированного цвета, подлежащего пересчету в соответствии с обратным преобразованием (1) и (4) в коды *RGB* и выводу на экран ЖК панели МФЦИ в условиях естественной освещенности:

$$\begin{bmatrix}R_{\text{кор}}\\G_{\text{кор}}\\B_{\text{кор}}\end{bmatrix}=\begin{bmatrix}X_r & X_g & X_b\\Y_r & Y_g & Y_b\\Z_r & Z_g & Z_b\end{bmatrix}^{-1}\begin{bmatrix}x_{\text{кор}}\frac{Y_{\text{кор}}}{y_{\text{кор}}}\\Y_{\text{кор}}\\\left(1-x_{\text{кор}}-y_{\text{кор}}\right)\frac{Y_{\text{кор}}}{y_{\text{кор}}}\end{bmatrix}. \quad (10)$$

Для осуществления расчетов по формулам (7)—(9) координаты цветности:

— ($x_{\text{пал}}$, $y_{\text{пал}}$, $Y_{\text{пал}}$) рассчитывают по кодам *RGB*, полученным в процессе светотехнических испытаний на АРМ для искусственного источника освещенности;

— ($x_{\text{иск}}$, $y_{\text{иск}}$, $Y_{\text{иск}}$) рассчитывают по данным таблицы для соответствующего вида искусственного источника света, компонента $Y_{\text{иск}}$ определяется по данным системы автоматической регулировки яркости (АРЯ) как относительная величина яркости (5) цвета, образованного диффузно отраженным искусственным светом, по отношению к собственной

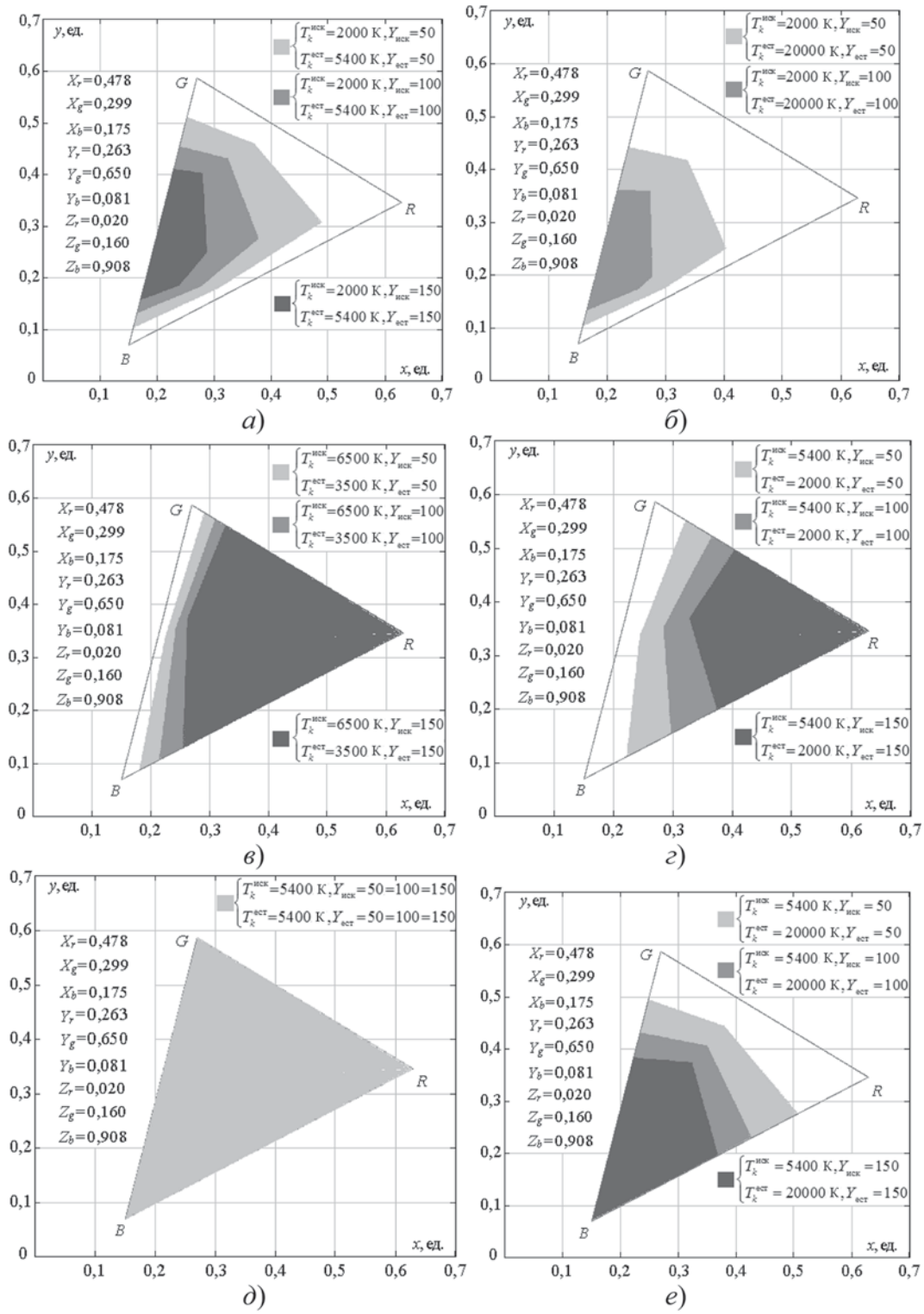


Рис. 5. Цветовой охват ЖК панели после коррекции воздействия внешней освещенности:

а — $T_k^{иск}$ = 2000 K, $T_k^{ест}$ = 5400 K; *б* — $T_k^{иск}$ = 2000 K, $T_k^{ест}$ = 2000 K; *в* — $T_k^{иск}$ = 6500 K, $T_k^{ест}$ = 3500 K; *г* — $T_k^{иск}$ = 5400 K, $T_k^{ест}$ = 2000 K; *д* — $T_k^{иск}$ = 5400 K, $T_k^{ест}$ = 5400 K; *е* — $T_k^{иск}$ = 5400 K, $T_k^{ест}$ = 20 000 K

яркости МФЦИ в цвете ($x_{пал}$, $y_{пал}$, $Y_{пал}$) при $E_{вн}$, т. е. $Y_{пал}:Y_{иск} = L_{и}:L_{вн}$;

— ($x_{ест}$, $y_{ест}$, $Y_{ест}$) рассчитывают по данным таблицы для соответствующего вида естественного источника света, компонента $Y_{ест}$ определяется по данным системы АРЯ как относительная величина яркости (5) цвета, образованного диффузно отраженным естественным светом, по отношению к собственной яркости МФЦИ в цвете ($x_{пал}$, $y_{пал}$, $Y_{пал}$) при $E_{вн}$, т. е. $Y_{пал}:Y_{ест} = L_{и}:L_{вн}$.

Соответствие вида источника естественной освещенности условиям эксплуатации летательного аппарата обеспечивается имеющейся на борту информацией:

— о географической широте и долготе полета;

— о времени суток в заданном часовом поясе полета;

— о высоте полета;

— о прогнозе погодных условий на маршруте полета и их коррекцией в реальном времени по данным бортовой метеолокационной станции.

Преобразования (7)—(9) были проверены на модельных данных, представленных в таблице, с использованием колориметрических характеристик естественных и искусственных источников света. Результаты моделирования в среде Mathcad 15.0 приведены на рис. 5.

Заполнение треугольника цветового охвата на рис. 5 выполнено координатами цветности, определенными для кодов *RGB*, которые принимают значение после коррекции (10) в допустимом диапазоне значений [0, 255].

Моделирование проводилось на персональном компьютере ASUS K56CB-X0391H со следующими характеристиками: процессор Intel(R) Core(TM) i5-3337U, 4 ядра, тактовая частота 1,8 ГГц, оперативная память 6 Гбайт под управлением операционной системы Windows 8.1. Профиль модели цветопередачи ЖК панели при моделировании задавался следующими значениями: $X_r = 0{,}478$; $X_g = 0{,}299$; $X_b = 0{,}175$; $Y_r = 0{,}263$; $Y_g = 0{,}650$; $Y_b = 0{,}081$; $Z_r = 0{,}020$; $Z_g = 0{,}160$; $Z_b = 0{,}908$.

Полученные на математических моделях теоретические результаты согласуются с ранее полученными результатами практических экспериментов, выполненных на образцах МФЦИ разработки ФГУП "Санкт-Петербургское опытно-конструкторское бюро "Электроавтоматика" им. П. А. Ефимова" в ГНИИИ ВМ МО РФ "Государственный научно-исследовательский испытательный центр авиационной медицины и военной эргономики".

Заключение

По результатам исследований можно сделать следующие выводы.

Цветовую палитру бортового средства индикации необходимо формировать на основе карты полей внешней освещенности в предполагаемой зоне полетов. Внешняя освещенность характеризуется спектральным составом излучения от естественного источника света (солнце, небо, луна, переотражения от облаков), которое имеет свою специфику в различных районах земной поверхности.

Цветовая температура естественного источника внешней освещенности варьируется от 2000 до 20 000 К и оказывает существенное влияние на воспринимаемый наблюдателем цвет изображения, индицируемого на ЖК панели МФЦИ. При разработке МФЦИ необходимо обеспечивать собственную яркость изделия на уровне $L_{и} \geqslant 1000$ кд/м^2 с коэффициентом диффузного отражения $\rho_{д} \leqslant 0{,}1$ %. Конструктивное размещение МФЦИ в кабине летательного аппарата должно исключать возможность зеркального отражения в направлении наблюдателя света от источника естественной освещенности. Зеркально отраженный свет от ЖК панели с коэффициентом зеркального отражения 1 % практически исключает возможность восприятия изображения при $E_{вн} = 75$ кЛк.

Предложенный способ коррекции позволяет выводить изображение на ЖК панель бортового средства индикации в условиях воздействия источников внешней освещенности с цветовой температурой в диапазоне 2000...20 000 К для ограниченного набора цветов цветовой палитры МФЦИ. Для формирования цветовой палитры может быть использована серия светотехнических измерений в лабораторных условиях при одном источнике искусственной освещенности с известными колориметрическими характеристиками.

Информация о зоне полетов летательного аппарата, необходимая для однозначного определения по формулам (7)—(10) компонент цветовой палитры, включает: географическую широту и долготу полета; высоту полета, поясное время полета; время захода и восхода солнца в этом временном поясе; погодные условия на маршруте полета. Такие данные передают летному составу перед вылетом.

Наблюдения за естественной внешней освещенностью на территории Российской Федерации на регулярной основе проводятся уже более 50 лет в метеорологической обсерватории МГУ им. М. В. Ломоносова. Имеющиеся результаты наблюдений могут быть положены в основу создания специальной карты полей и высот естественной освещенности в зонах полетов летательных аппаратов в различных погодных условиях.

Список литературы

1. **Жаринов И. О., Жаринов О. О.** Бортовые средства отображения информации на плоских жидкокристаллических панелях: Учеб. пособие. Информационно-управляющие системы. СПб.: ГУАП, 2005. 144 с.
2. **Греф П., Хульце Х. Г.** Технологии адаптивного изменения яркости задней подсветки телевизионных ЖК-экранов // Электронные компоненты. 2008. № 3. С. 70—76.
3. **Дятлов В. М.** Разработка и исследование конструкции стеклопакета жидкокристаллического экрана // Военная электроника и электротехника. Труды 22 ЦНИИИ Минобороны России. 2010. Вып. 62. С. 270—279.
4. **Высоцкий В., Бауткин В.** Улучшение оптических свойств жидкокристаллических панелей // Современная электроника. 2009. № 8. С. 22—25.
5. **Зайцев А.** Требования и испытания TFT-модулей NEC Electronics, работающих в жестких условиях эксплуатации // Компоненты и технологии. 2007. № 7. С. 16—20.

6. **Индутный И. З., Шепелявый П. Е., Михайловская Е. В.** и др. Градиентные светопоглощающие покрытия SiO_x—Me для дисплейных экранов // Журнал технической физики. 2002. Т. 72, № 6. С. 67—72.
7. **Костишин М. О., Жаринов И. О., Жаринов О. О.** Исследование визуальных характеристик средств отображения пилотажно-навигационных параметров и геоинформационных данных в авионике // Информационно-управляющие системы. 2014. № 4. С. 61—67.
8. **Синяк М.** Влияние внешнего освещения на принятие оценочного решения о качестве полиграфических оттисков // КомпьюАрт. 2008. № 5. С. 38—45.
9. **Синяк М.** Цвет как критерий оценки // Мир этики. 2006. № 3. С. 52—56.
10. **Белов Н. П., Яськов А. Д., Герасимов В. Н.** Лабораторный спектрометр для исследования коэффициента отражения и определения параметров цветности диффузно отражающих поверхностей // Известия вузов. Приборостроение. 2010. Т. 53, № 7. С. 74—78.
11. **Жаринов И. О., Жаринов О. О.** Исследование распределения оценки разрешающей способности преобразования Грассмана в системах кодирования цвета, применяемых в авионике // Программная инженерия. 2014. № 8. С. 40—47.
12. **Гарютин И. А.** Формирование критерия подобия цветовых характеристик газоразрядных металлогалогенных ламп // Известия вузов. Приборостроение. 2013. Т. 56, № 3. С. 71—75.
13. **Костишин М. О., Жаринов И. О.** Исследование оптических параметров бортовых средств индикации геоинформационных данных // Вестник Череповецкого государственного университета. 2014. № 2. С. 5—9.
14. **Жаринов И. О., Жаринов О. О., Парамонов П. П.** и др. Принципы построения автоматических систем в канале управления тепловыми и светотехническими характеристиками бортовых средств индикации // Известия вузов. Приборостроение. 2014. Т. 57, № 12. С. 34—38.

I. O. Zharinov, Associate Professor, Chief of Department, Saint Petersburg National Research University of Information Technologies, Mechanics and Optics (University ITMO), Chief of Learning-Scientists Center, SPb Scientific Design Bureau "Electroavtomatica" n. a. P. A. Efimov, e-mail: igor_rabota@pisem.net, **O. O. Zharinov,** Associate Professor, Saint Petersburg State University of Aerospace Equipment

Research of Influence of the External Illuminance on the Color Measurement Characteristics for Visually Percepted Image in Avionics Engineering

The problem is considered and research is performed for the software-based correction method dealing with digital codes of components of color palette that is using in on-board indication equipment. It is shown that color palette, obtained with the use of any given source of external illuminance that is mounted in automated workstation, can't be successfully used for all possible operating conditions when color temperature of external varying widely (from 2000 to 20 000 Kelvins). In order to display an image in varying conditions of external illuminance falling on the screen there was proposed the method and formulas allowing to correct digital codes of any given color palette. It is shown, that relations between brightness value of an image on LCD-panel and brightness of the color, caused by diffuse reflection of the light of external illuminance sourse, determine not only technique for calculation of the contrast value for any given image element, but also restricts color gamut, that person can visually percept.

Keywords: *indication, avionics, chromaticity coordinates, external illuminance, correction*

References

1. **Zharinov I. O., Zharinov O. O.** *Bortovye sredstva otobrazhenija informacii na ploskih zhidkokristallicheskih paneljah* (Onboard Display on Flat Liquid Crystal Panels), Saint Petersburg, Informacionno-upravljajushhie sistemy, 2005, 144 p. (in Russian).
2. **Gref P., Hul'ce H. G.** Tekhnologii adaptivnogo izmeneniya yarkosti zadnei podsvetki televizionnykh ZhK-ekranov. *Jelektronnye komponenty*, 2008, no. 3, pp. 70—76 (in Russian).
3. **Djatlov V. M.** Razrabotka i issledovanie konstruktsii steklopaketa zhidkokristallicheskogo ekrana. *Voennaja jelektronika i jelektrotehnika,* 2010, vol. 62, pp. 270—279 (in Russian).
4. **Vysockij V., Bautkin V.** Uluchshenie opticheskikh svoistv zhidkokristallicheskikh panelei. *Sovremennaja jelektronika*, 2009, no. 8, pp. 22—25 (in Russian).
5. **Zajcev A.** Trebovaniya i ispytaniya TFT-modulei NEC Electronics, rabotayushchikh v zhestkikh usloviyakh ekspluatatsii. *Komponenty i tehnologii*, 2007, no. 7, pp. 16—20 (in Russian).
6. **Indutnyj I. Z., Shepeljavyj P. E., Mihajlovskaja E. V., Park Ch. V., Li Dzh. B., Do Ja. R.** Gradientnye svetopogloshchayushchie pokrytiya SiO_x—Me dlya displeinykh ekranov. *Zhurnal tehnicheskoj fiziki*, 2002, vol. 72, no. 6, pp. 67—72 (in Russian).
7. **Kostishin M. O., Zharinov I. O., Zharinov O. O.** Issledovanie vizual'nykh kharakteristik sredstv otobrazheniya pilotazhno-navigatsionnykh parametrov i geoinformatsionnykh dannykh v avionike. *Informacionno-upravljajushhie sistemy*, 2014, no. 4, pp. 61—67 (in Russian).
8. **Sinjak M.** Vliyanie vneshnego osveshcheniya na prinyatie otsenochnogo resheniya o kachestve poligraficheskikh ottiskov. *Komp'juArt*, 2008, no. 5, pp. 38—45 (in Russian).
9. **Sinjak M.** Tsvet kak kriterii otsenki. *Mir jetiki*, 2006, no. 3, pp. 52—56 (in Russian).
10. **Belov N. P., Jas'kov A. D., Gerasimov V. N.** Laboratornyi spektrometr dlya issledovaniya koeffitsienta otrazheniya i opredeleniya parametrov tsvetnosti diffuzno otrazhayushchikh poverkhnostei. *Izvestija vuzov. Priborostroenie*, 2010, vol. 53, no. 7, pp. 74—78 (in Russian).
11. **Zharinov I. O., Zharinov O. O.** Issledovanie raspredeleniya otsenki razreshayushchei sposobnosti preobrazovaniya Grassmana v sistemakh kodirovaniya tsveta, primenyaemykh v avionike. *Programmnaya ingeneria*, 2014, no. 8, pp. 40—47 (in Russian).
12. **Garjutin I. A.** Formirovanie kriteriya podobiya tsvetovykh kharakteristik gazorazryadnykh metallogalogennykh lamp. *Izvestija vuzov. Priborostroenie*, 2013, vol. 56, no. 3, pp. 71—75 (in Russian).
13. **Kostishin M. O., Zharinov I. O.** Issledovanie opticheskikh parametrov bortovykh sredstv indikatsii geoinformatsionnykh dannykh. *Vestnik Cherepoveckogo gosudarstvennogo universiteta*, 2014, no. 2, pp. 5—9 (in Russian).
14. **Zharinov I. O., Zharinov O. O., Paramonov P. P., Kostishin M. O., Sudarchikov S. A.** Printsipy postroeniya avtomaticheskikh sistem v kanale upravleniya teplovymi i svetotekhnicheskimi kharakteristikami bortovykh sredstv indikatsii. *Izvestija vuzov. Priborostroenie*, 2014, vol. 57, no. 12, pp. 34—38 (in Russian).

УДК 338.364.2

Д. С. Пащенко, канд. техн. наук, MBA, независимый консультант в области разработки программного обеспечения, e-mail: denpas@rambler.ru, г. Москва

Изменения в процессе производства программного обеспечения: исследование в Центральной и Восточной Европе

Авторское исследование по методу Дельфийской панели, результаты которого представлены в статье, проведено в IT-компаниях в середине 2014 г. В нем приняли участие 78 IT-инженеров из 11 стран Центральной и Восточной Европы, включая Россию. Представлены полные результаты исследования, а именно: основные проблемы и риски управления изменениями; опыт специалистов в сфере производственных изменений; важность стандартизации процессных моделей в IT-компаниях; особенности эмоционального восприятия перечисленных изменений инженерами. Анализ результатов опросов экспертов определил особенности организационного сопротивления изменениям в IT-отрасли, необходимость своевременного планирования и закрепления изменений в производственной практике, аспекты повышения прогнозируемости влияния изменений на проектную деятельность. Автор и эксперты подготовили набор развернутых рекомендаций по уменьшению влияния типичных рисков на успешный характер проведения производственных изменений на каждом этапе программного проекта.

Ключевые слова: *производственные изменения, разработка программных продуктов, организационное сопротивление в софтверной компании, вовлечение сотрудников в управление изменениями, закрепление изменений в производственной практике*

Введение

Страны Центральной и Восточной Европы (включая Россию, Украину и Беларусь) в последние 20 лет прошли схожий путь эволюции процессов разработки программного обеспечения (ПО). Такой путь начинался от постсоветских стандартов на разработку и закончился масштабным влиянием транснациональных корпораций, предъявляющих достаточно строгие требования к параметрам разрабатываемого ими ПО. При этом сам рынок коммерческого ПО в отдельной стране также прошел схожий путь развития: в сегменте b2b существенную долю (особенно в области бизнес-приложений) занимают локальные поставщики или поставщики из соседних стран; в сегменте b2c доминируют зарубежные разработчики (в основном, американские). При этом первая волна собственников IT-компаний уже ушла или уходит на пенсию, передавая управление профессиональным менеджерам, а сделки по слиянию и поглощению компаний стали частью этого бизнеса. Очевидно, в течение такой 20-летней истории развития отрасль прошла набор технологических революций, каждая из которых нуждалась в управлении значительными изменениями в производстве в компаниях, занимающихся разработкой ПО.

Управлению изменениями в производстве программного продукта посвящено много исследовательских работ [1—3], а сама история вопроса уходит уже в XIX век. Однако, в силу обладающих спецификой этой сферы деятельности ментальных и трудовых факторов, управление изменениями в IT-компаниях имеет набор существенных особенностей. В исследовании, результаты которого представлены далее, рассмотрены этапы внедрения изменений в производственную модель софтверной компании: от этапа планирования и подготовки команд разработчиков до закрепления результатов в производственной практике.

В проведенном исследовании было получено обобщенное мнение и данные о реальном опыте участия в проведении производственных изменений от 78 IT-инженеров из 11 стран Центральной и Восточной Европы. Выбор этого макрорегиона, как уже было упомянуто ранее, обусловлен схожим сценарием 20-летнего развития процессов производства ПО и в целом бизнеса в области информационных технологий. Условно можно разделить этот эволюционный путь на период стандартов постсоветской эпохи и период движения навстречу западноевропейским и американским стандартам качества и процессного управления в софтверном производстве. Второй период начался с середины 1990-х гг. и не завершен до сих пор. Несмотря на географическую близость и экономическую интеграцию центральноевропейских стран в Европейский Союз, в целом уровень процессного развития IT-компаний, подтвержденный сертификатами CMMI (Capability Maturity Model Integration), примерно одинаков в Центральной и Восточной Европе [4]. Ис-

ключениями можно назвать Польшу (так как польские IT-услуги и продукты востребованы и в Германии, и в Украине) и Турцию, где число сертифицированных компаний в 2012—2014 гг. по CMMI в различных моделях велико, а их совокупное влияние на европейский или мировой рынок ПО незначительно. В связи с последним обстоятельством Турция на момент проведения исследования представлялась страной с несхожим в регионе вектором развития моделей зрелости производственных процессов и была исключена из данного исследования.

Исследование проводилось по методу Дельфийской панели [5] и было направлено на получение максимально согласованного мнения участвующих в нем экспертов. В исследовании было выделено несколько секций, каждая из которых относилась к определенному этапу внедрения производственных изменений, ранее описанных автором в исследовании [6]. К таким этапам относятся: планирование изменений; подготовка коллектива; детализация изменений; внедрение изменений в производственную практику. В части внедрения были предусмотрены вопросы, относящиеся также к закреплению изменений в практике компании и анализу полученных результатов.

В исследовании основное внимание было акцентировано на специфических рисках неблагоприятных событий (далее — рисках) и особенностях, сопровождающих внедрение производственных изменений в IT-компаниях на каждом этапе реализации их внутреннего проекта. К числу таких особенностей относят:

- роль руководителя проекта;
- организационное сопротивление коллектива изменениям процесса производства;
- вовлечение сотрудников в управление изменениями;
- противоречия целей внедряемых изменений и производственных целей проекта;
- восприятие инженерами организационных мер поддержки внедрения и закрепления изменений.

В статье также представлены рекомендации по преодолению типичных рисков и издержек, сопутствующих внутренним проектам улучшения процессов производства, которые направлены на баланс усилий различных структур и ролей в компании для наименее затратного по ресурсам и комфортного для коллектива внедрения изменений в производственные процессы создания ПО. Отметим также, что зрелость производственных процессов напрямую связана с качеством ПО и итоговым коммерческим успехом компаний [7].

Методология и процесс исследования

Исследование проводилось с апреля по июль 2014 г. по методу Дельфийской панели в три этапа, в нем приняли участие 78 экспертов из Центральной и Восточной Европы, включая Россию, Украину и Беларусь. Все участники исследований — реальные инженеры — члены проектных команд разработчиков ПО (аналитики, разработчики, архитекторы) с большим опытом работы в IT-отрасли. Почти все участники исследований имели опыт успешных программных проектов и представляли ведущие IT-компании своих стран.

На этапе 1 экспертам был предложен список вопросов, разделенных на секции, двух типов:

- с одним возможным вариантом ответа;
- с несколькими вариантами ответа.

Именно по данным ключевым вопросам проводилось исследование мнения экспертов.

На этапе 2 мнение каждого эксперта по каждому вопросу сравнивалось с доминирующим мнением по результатам опроса всех участвующих экспертов. У каждого эксперта была возможность скорректировать свое мнение или прокомментировать доминирующий ответ, указав особенности своей точки зрения.

На этапе 3 были подведены итоги исследования, запрашивалась дополнительная и уточняющая информация.

Для вопросов с одним вариантом ответа были построены столбчатые диаграммы, демонстрирующие распределение мнений после этапа 2. Для вопросов с несколькими вариантами ответа были построены списки, ранжированные по популярности каждого ответа.

На этапе 2 было отмечено традиционное снижение активности экспертов, оставшееся, впрочем, в рамках допущений метода Дельфийской панели. В таблице приведены данные об активности экспертов по соответствующим этапам.

Активность экспертов, участвующих в исследовании, по этапам

Эксперты	Этап исследования		
	1	2	3
Число экспертов	78	61	78
Ответившие эксперты, %	100	78	100

Далее приведены данные по экспертам, участвующим в исследовании, позволяющие оценить авторитетность собранных мнений, представленный ими опыт и географию региона исследования.

Приведенный по результатам исследования опыт чаще всего является релевантным в схожих типах бизнеса в IT-компаниях. Типы производства ПО представлены экспертами следующим образом:

- 9 % экспертов представили свой опыт в разработке для собственных нужд компании (*in-house*);
- 11 % экспертов представили свой опыт, приобретенный в системных интеграторах;
- 31 % экспертов представили свой опыт, приобретенный в вендорах ПО (ISV);
- 49 % экспертов представили свой опыт деятельности в компаниях, выполняющих заказную разработку ПО (в том числе *out-source*).

География экспертов представлена следующим образом:

- 46 % экспертов из России и Беларуси;
- 26 % экспертов из Балканского субрегиона (Сербия, Босния, Болгария, Молдова);
- 15 % экспертов из Центральной Европы (Чехия, Словакия, Венгрия);
- 13 % экспертов из Польши и Украины.

По возрасту эксперты в основном представляли среднюю группу, ассоциирующуюся в отрасли информационных технологий с расцветом творческого потенциала для команд разработки:

- 31 % экспертов возрастного диапазона 20...29 лет;
- 64 % экспертов возрастного диапазона 30...39 лет;
- 5 % экспертов возрастного диапазона 40...49 лет;
- 0 % экспертов возрастного диапазона 50 лет и более.

При этом подавляющее число экспертов работали в области разработки ПО значительное время. Так, среди

участников исследования весьма мало экспертов, работавших в области информационных технологий менее 5 лет:

- 7 % экспертов работали в области разработки ПО 2...5 лет;
- 44 % экспертов работали в области разработки ПО 5...10 лет;
- 49 % экспертов работали в области разработки ПО более 10 лет;
- 0 % экспертов работали в области разработки ПО менее 2 лет.

Результаты исследования

Вопросы исследования были разделены на несколько секций, включая секцию общих вопросов и следующие секции, описывающие фазы внедрения изменений в производство ПО в IT-компании:

1) планирование и подготовка к внедрению изменений в производственные процессы разработки ПО;

2) внедрение изменений в процессы разработки ПО;

3) закрепление изменений и анализ результатов.

Результаты исследования представлены в форме диаграмм с мнениями экспертов для вопросов, подразумевающих только один вариант ответа. Для вопросов, где возможно было дать несколько вариантов ответов, приведены наиболее популярные ответы.

Общие вопросы

В секции "Общие вопросы" эксперты определяли основную роль стандартизации производственных процессов и ее влияние на основные аспекты бизнеса, включая степень успеха разработки ПО, качество программных продуктов, наем специалистов.

Большинство экспертов согласилось, что стандартизация процессов имеет прямое влияние на высокий уровень конечного качества программных продуктов. По мнению некоторых экспертов, стандартизация также может временно снижать уровень качества в силу внесения дополнительных возмущений в текущие бизнес-процессы в течение какого-то ограниченного времени. На рис. 1 приведены мнения экспертов (здесь и далее — в процентах экспертов от общего состава участвующих в исследовании).

Эксперты также выделили исключительные случаи, когда продукты высокого качества создают с помощью условно повторяемых процессов небольшие постоянные по составу команды профессионалов без явной стандартизации и формального описания производственных процессов.

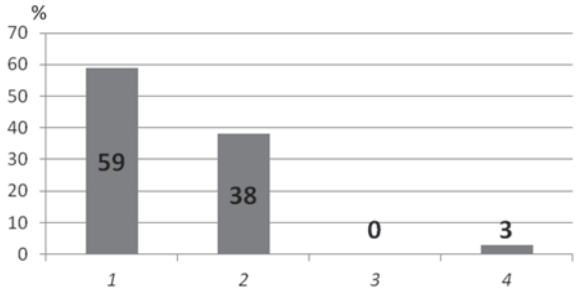


Рис. 1. Влияет ли общая стандартизация процессов производства ПО на уровне всей компании на конечное качество получаемого программного продукта:

1 — всегда влияет; *2* — часто влияет; *3* — иногда влияет; *4* — почти не влияет

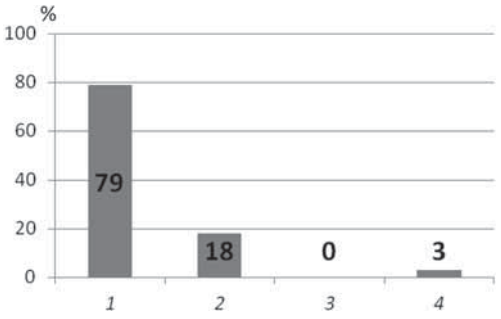


Рис. 2. Должен ли процесс подготовки и внедрения значительных изменений в производство ПО быть прозрачным для команд разработки:

1 — да, с начала, с этапа планирования; *2* — да, позже, с этапа непосредственного внедрения; *3* — нет, это неважно; *4* — затрудняюсь ответить

Внедрение изменений в производственные процессы в IT-компаниях растянуто во времени и раньше или позже, но требует вовлечения в сферу этой деятельности инженеров. Эксперты определили необходимость режима прозрачности данного процесса для разработчиков с самых ранних этапов выполнения проекта (рис. 2). Вместе с тем вовлечение большого числа специалистов (особенно с низкой мотивацией и квалификацией) наоборот затрудняет внедрение изменений и увеличивает его сроки и издержки.

Таким образом, инженеры-эксперты считают необходимым раннее информирование команд о будущих процессных изменениях. Однако уровень детализации такого информирования зависит от особенностей каждого коллектива и масштаба изменений.

Стандартизация процессов и производственная зрелость обладают эмоциональным эффектом. В том числе это касается и области продаж продуктов и услуг в IT-бизнесе, не говоря уже о явных положительных факторах для повышения качества продуктов и эффективности бизнеса [8]. Интересным представляется оценка влияния этого эмоционального фактора на привлечение сотрудников и повышение их профессиональной мотивации (рис. 3).

По представленным данным можно сделать вывод, что инженеры-эксперты довольно скептически относятся к влиянию декларируемой зрелости производства на по-

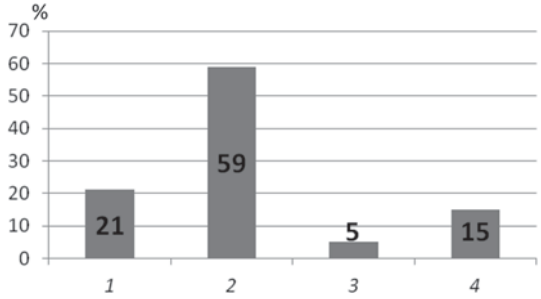


Рис. 3. Эффективно ли декларирование стандартизированных процессов производства ПО и наличие сертификатов при привлечении талантливых инженеров на работу в компанию:

1 — да, помогает привлечь сотрудников; *2* — влияния на привлечение специалистов не оказывает; *3* — скорее отпугивает талантливых инженеров; *4* — затрудняюсь ответить

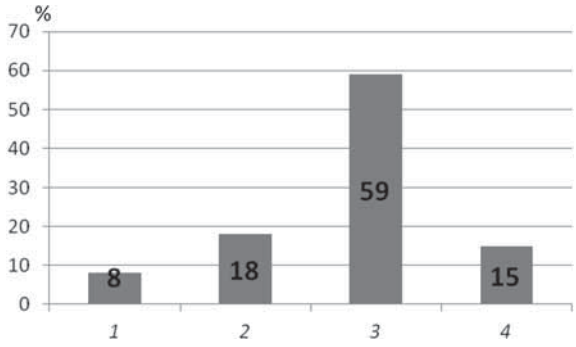


Рис. 4. Является ли работа в компании со стандартизированными процессами предметом особенной гордости для рядовых инженеров и аналитиков:

1 — почти всегда; *2* — довольно часто; *3* — иногда является; *4* — почти не является

вышение притягательности рабочего места в компании. Такое отношение характерно как для вновь привлекаемых сотрудников, так и для тех, кто уже в компании работает. Эксперты отметили (рис. 4), что лишь иногда члены команд разработки испытывают особенную гордость от факта, что работают в компании со зрелыми процессами разработки ПО.

Планирование и подготовка к внедрению изменений в производственные процессы разработки ПО

В данной секции эксперты оценили основные действия, которые необходимы для подготовки к внедрению изменений исходя из опыта и практики своих компаний. В этом смысле первый этап планирования, в результате которого появляется некоторый обзор предлагаемых изменений производственного процесса, должен обеспечить вовлечение в его обсуждение наиболее заинтересованных сотрудников компании. При таком подходе на следующем этапе подготовки коллектива к изменениям уже сложившаяся инициативная команда осуществляет формальное и неформальное информирование всех сотрудников о предстоящих нововведениях.

Эксперты не смогли найти зависимостей между регулярностью процесса внедрения изменений в производство и усилиями инициаторов на уровне как всей компании, так и отдельных выполняемых ею проектов (рис. 5).

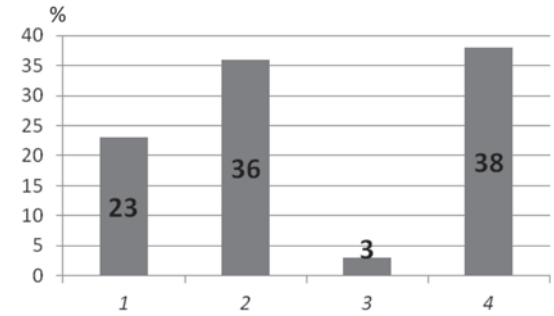


Рис. 5. Имеет ли процесс внедрения изменений в производство ПО явную регулярность:

1 — да, на уровне всей компании; *2* — да, на уровне каждого проекта; *3* — нет, изменения вносятся спонтанно; *4* — часть изменений запланирована, часть вносится спонтанно

Обычно часть таких изменений появляется спонтанно, а централизованные усилия по их планированию могут оставаться незамеченными.

Недостаточный авторитет, непрозрачный характер деятельности, а иногда и невостребованность в IT-компаниях формальных структур (таких как Software Process Engineering Group (SEPG), Офис Управления Проектами, Офис процессного развития) размывает восприятие постоянных улучшений процессов в производстве как одного из ключевых факторов обеспечения качества программных продуктов. В ощущениях и опыте инженеров-экспертов, демонстрируемых в данном исследовании, роль централизованного внесения изменений в производство кажется недооцененной. При этом на первый план выходят проектные инициативы.

Обобщенная оценка мнения экспертов, участвующих в исследованиях, показала, что проектный менеджер (ПМ) является ключевой фигурой в инициировании производственных изменений (рис. 6). При этом следует отметить и некоторые отклонения от такой оценки. Например, в проектных командах, работающих по гибким методологиям (или классическому Microsoft Solution Framework), роль ПМ не очень выражена. По этой причине любой член проектной команды может являться инициатором изменений в производственных процессах. Кроме того, в компаниях с развитым централизованным процессным управлением в составе Дирекции качества (Офиса управления проектами, SEPG и других структурах) ПМ могут быть частично вовлечены в деятельность такой структуры.

Представленные результаты показывают, что в большой части софтверных компаний вертикальные коммуникационные линии от директоров к инженерам, относящиеся к развитию производственных процессов, проходят мимо соответствующих формальных корпоративных структур. В восприятии инженеров ПМ (руководители проектов) являются основными инициаторами изменений. С одной стороны, этот факт означает, что бизнес-цели проектов и цели изменений могут быть сопряжены лучше, так как управление ими пересекается в одном инициативном лице. С другой стороны, поддержка изменений руководителем проекта становится одним из ключевых факторов.

Эксперты отметили (рис. 7), что по их опыту обычно члены проектных команд не имеют достаточно много времени для подготовки к значительным изменениям

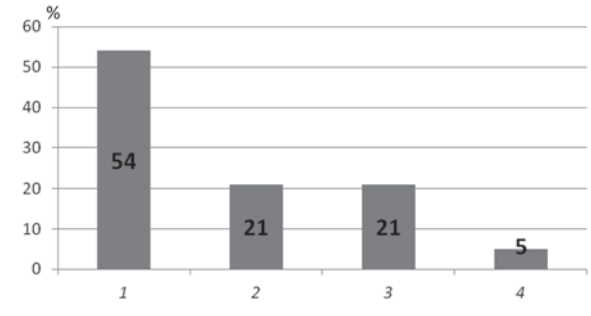


Рис. 6. Кто обычно инициирует внедрение изменений в производственные процессы разработки ПО:

1 — проектный менеджер; *2* — дирекция качества/процессного развития; *3* — члены проектных команд; *4* — первое лицо компании/дирекции разработки

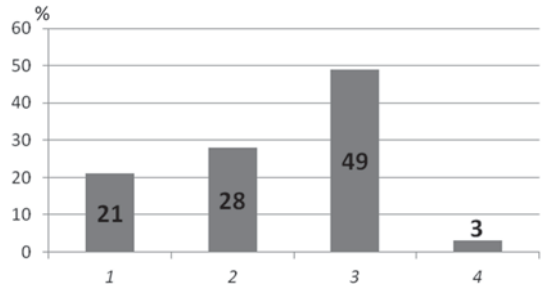


Рис. 7. Когда обычно о внедрении изменений в производственные процессы разработки ПО становится известно членам проектных команд:

1 — за месяцы до начала внедрения; *2* — за недели до начала внедрения; *3* — в начале внедрения изменений; *4* — уже после официальной даты запуска внедрения изменений

в производственных процессах. При этом, по опыту почти половины экспертов, проектные команды получают информацию о предстоящих изменениях только в начале их реализации. Такая ситуация демонстрирует недостаточность усилий по раннему информированию сотрудников со стороны менеджмента, а также подчеркивает осознанную необходимость инженеров в получении информации о предстоящих изменениях заранее.

Среди эффективных и распространенных методов подготовки сотрудников к производственным изменениям эксперты выделили:

- информирование сотрудников (отметили почти 70 % экспертов);
- привлечение инженеров и аналитиков к планированию изменений (отметили 65 % экспертов).

При этом следует учесть разную степень собственной готовности специалистов к подобной активности. Привлечение инженеров к планированию изменений требует как взвешенного анализа персонально-профессиональных качеств вовлекаемого в планирование сотрудника, так и оценки влияния такого вовлечения на производственные проекты компании. Дополнительная нагрузка, ведущая к переработкам и увеличению рабочего дня вовлекаемых инженеров, не скажется положительно на планировании изменений.

Эксперты также оценили необходимые параметры буферного срока подготовки команд разработки к внедрению изменений в производственных процессах. Более половины экспертов (55 %) по своему опыту оценили, что такой период занимает несколько недель. При этом только 25 % встречали случаи внедрения изменений, когда командам разработки давали менее недели на подготовку.

Целесообразным представляется в течение данного буферного срока осуществлять процедуры по подготовке коллектива к изменениям. Среди таких мер одной из наиболее естественных и простых является информирование сотрудников о предстоящих изменениях.

Эксперты, участвующие в исследовании, определили следующие наиболее популярные типы такого информирования:

- линейные и проектные руководители проводят встречи и разъяснения (встречалось в практике 99 % экспертов);
- изменения анонсируются первым лицом компании (дирекции) (встречалось в практике 30 % экспертов).

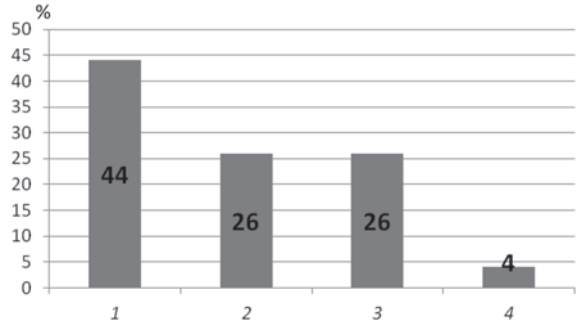


Рис. 8. Кто несет наибольшую персональную ответственность за успех внедрения изменений в процессы производства ПО:

1 — каждый руководитель проекта производства ПО; *2* — только руководитель всего производства в компании; *3* — все члены проектных команд; *4* — инициатор изменений вне зависимости от его должности

Причинно-следственная связь крайне важна в формировании отношения каждого инженера к изменениям [9]. Исследование показало наиболее распространенные причины внесения изменений в производственные процессы. К их числу относятся:

- объективная необходимость изменений (встречалось в практике 64 % экспертов);
- следование требованиям заказчика, аудиторов, ожиданиям рынка (встречалось в практике 62 % экспертов).

Таким образом, именно экономические мотивы являются наиболее распространенными в восприятии инженеров с точки зрения объяснения внедряемых изменений в производстве. Из этого следует, что при обосновании для проектных команд необходимости внедрения изменений инициативной группой должны использоваться экономические факторы и выгоды от успешного внедрения и закрепления изменений.

Несмотря на различия в административно-организационной иерархии, в командах, работающих по классическим итерационным и гибким методологиям, руководитель проекта, по мнению экспертов, несет максимальную личную ответственность за успех внедрения изменений в процессы разработки ПО (рис. 8).

Полученный результат еще раз подчеркивает важную роль руководителей проектов, являющихся ключевыми фигурами как с точки зрения решения бизнес-задач в производственных проектах, так и с точки зрения управления изменениями. Полученный результат исследования может быть использован в качестве аргумента при вовлечении руководителей проектов в управление производственными изменениями на корпоративном уровне.

Внедрение изменений в процессы разработки ПО

В секции на направлении внедрения изменений в процессы разработки ПО эксперты выделили основные вопросы, связанные с непосредственным внедрением в практику запланированных изменений, а также типичные риски и необходимый выбор приоритетов в данном процессе.

Непосредственное внедрение и закрепление изменений является наиболее трудоемким и эмоционально напряженным этапом, который сопряжен с появлением ранее неучтенных факторов риска. При этом растягивание во времени данного этапа может привести к откатам назад в совершенствовании производственных процессов и даже существенным сбоям в ритме производства в будущем.

Эксперты выделили наиболее востребованные организационные мероприятия при внедрении изменений в процессы производства ПО. К их числу относятся:

- устные распоряжения и контроль со стороны проектного менеджера (встречалось в практике 59 % экспертов);
- издание приказов, распоряжений, изменений бизнес-процессов (встречалось в практике 57 % экспертов).

Таким образом, исполнение устных распоряжений менеджера проекта, вовлеченного в управление изменениями, должно быть поддержано с помощью централизованных и формализованных письменных приказов и изменений в бизнес-процессах. Тем более, что подобные изменения бизнес-процессов могут сопровождаться средствами их автоматизации и настройкой соответствующих систем и механизмов.

При этом эмоциональная составляющая в части внедрения изменений является крайне важной. Нередко именно она определяет успех закрепления изменений в практике [10]. По этой причине устные распоряжения и личный контроль непосредственных руководителей, пользующихся авторитетом у сотрудников, не могут заменить механически рассылаемые по электронной почте письма.

Безусловно, автоматизация технологических операций при разработке ПО — это один из действенных методов внедрения обязательного исполнения измененных процессов. Сообщество экспертов, участвующих в исследовании, исходя из своего опыта подтвердило данное утверждение (рис. 9).

Эксперты определили список существенных трудностей, которые являются типичными при управлении изменениями и при стандартизации производственного процесса IT-компании. Прежде всего, это формальное внедрение без результатов и понимания целей (встречалось в практике 77 % экспертов) и конфликты между целями проекта и целями внедрения изменений (встречалось в практике 54 % экспертов). Эксперты отметили также существенные риски, сопровождающие внедрение изменений в производство в большинстве IT-компаний (рис. 10).

Данные трудности в восприятии и инженеров, и IT-менеджеров являются одинаково актуальными, что подтверждается более ранним авторским исследованием [6]. Безусловно, каждый такой существенный вопрос и риск неблагоприятного результата требуют учета, анализа и мер реагирования.

Эксперты рассмотрели набор организационных мер, направленных на работу со специфическими рисками неблагоприятных событий при управлении изменениями в IT-отрасли, отмеченными на рис. 10. Так, эксперты выделили несколько эффективных мер для преодоления организационного сопротивления, в их числе:

- разъяснительная работа с элементами подавления;
- вовлечение сопротивляющихся во внедрение изменений;
- положительная мотивация к принятию изменений.

Мотивация команд разработчиков — это ключевой фактор в управлении изменениями. Эксперты определили наиболее востребованные методы мотивации для поддержки внедрения изменений в производственные процессы создания ПО. К их числу относятся:

- воодушевление и поощрение использования новых практик (встречалось в практике 82 % экспертов);
- общественное порицание за уклонение от следования внедренным стандартам (встречалось в практике 31 % экспертов).

Денежные штрафы и поощрения оказались куда менее популярными методами мотивации сотрудников. Этот факт означает, что в восприятии инженеров данные методы не приведут к следованию измененным бизнес-процессам и поддержке изменений.

В теории управления изменениями известен интересный конфликт целей между основным бизнес-процессом и частной целью внедрения изменений в данный бизнес-процесс. Эксперты, участвующие в исследовании, не сочли распространенным случай, когда цели изменений могут быть приоритетнее, чем текущая деятельность по производству продукта проекта (рис. 11). Однако некоторые эксперты выделили исключительные случаи, когда

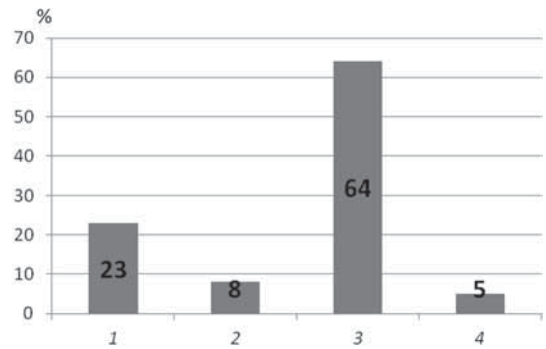


Рис. 9. Как подкрепляется внедрение изменений в производственные процессы автоматизацией технологических процессов разработки:

1 — никак не связаны между собой; *2* — автоматизация процессов позволяет игнорировать изменения; *3* — автоматизация процессов заставляет следовать изменениям; *4* — автоматизацию технологических процессов можно умышленно обойти

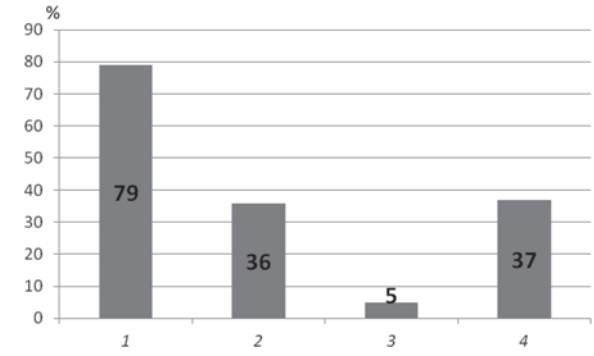


Рис. 10. Какие риски обычно сопровождают внедрение изменений в процессы производства ПО:

1 — резкое падение качества ПО и сроков поставки релизов; *2* — падение мотивации сотрудников проектных команд; *3* — падение авторитета проектного и/или линейного руководителя; *4* — общее возрастание конфликтности в проектной команде

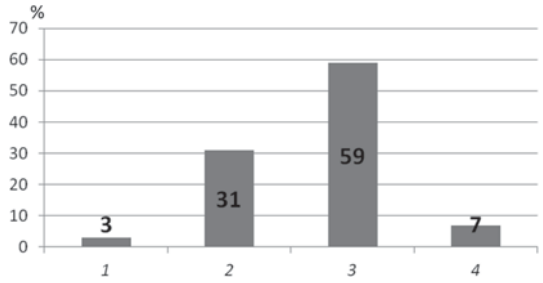


Рис. 11. Насколько часто цели внедрения изменений могут быть приоритетнее, чем текущая деятельность по производству продукта проекта:

1 — очень часто; *2* — часто; *3* — редко; *4* — никогда

прямые распоряжения топ-менеджмента делали изменения в процессах более приоритетными, чем основные цели проектов, в которые внедрялись изменения.

Эксперты также определили наиболее типичные издержки, которые несет проект в течение времени внедрения изменений в производственные процессы:

- издержки по качеству и/или срокам поставки продукта (встречалось в практике 85 % экспертов);
- ухудшение внутреннего климата в команде проекта (встречалось в практике 31 % экспертов);
- уход части специалистов из компании/проекта (встречалось в практике 27 % экспертов).

При этом в подавляющем большинстве программных проектов отношения с заказчиком не претерпевали ухудшений при изменении внутренних производственных процессов.

Представленный выше список издержек имеет прямое влияние на экономическую составляющую производственных проектов. Такие издержки должны быть учтены в плане рисков и в рабочем плане. Должны быть разработаны соответствующие предупреждающие и корректирующие воздействия. Очевидно, что реализация программы предупреждения рисков — это длительный процесс, который следует учитывать еще на этапе планирования изменений.

Закрепление изменений и анализ результатов

Внедрение изменений в производственные процессы компании требует значительных усилий на всех ее уровнях. В секции "Закрепление изменений и анализ результатов" эксперты высказали свое мнение и поделились опытом в части эффективности применяемых мер закрепления изменений в производственной практике. Эксперты выделили типичные организационные меры для закрепления результатов внедренных изменений в производстве на уровне проекта. В их числе:

- аудит и внимание со стороны руководителя проекта (встречалось в практике 69 % экспертов);
- автоматизация внедренных изменений (встречалось в практике 56 % экспертов);
- закрепление в документах проекта (встречалось в практике 49 % экспертов);
- поощрение использования новой практики (встречалось в практике 37 % экспертов).

Эти показатели вновь иллюстрируют важность роли руководителя проекта в управлении изменениями, хотя в производственных проектах он имеет ряд других важных обязанностей, направленных на решение основных бизнес-задач проекта.

Реализация новых подходов в производстве программного продукта требует регулярного мониторинга и контроля исполнения работ в соответствии с принятым планом. Эксперты определили востребованность ряда мер контроля исполнения командами разработчиков новых стандартов производства ПО, в том числе:

- аудит со стороны руководителя проекта (встречалось в практике 60 % экспертов);
- разбор инцидентов после сбоев в качестве продукта проекта (встречалось в практике 55 % экспертов).

Между тем для некоторых экспертов идея аудита со стороны руководителя проекта кажется странной. Они считают, что основное внимание в его деятельности должно быть направлено на достижение целей самого проекта без глубокого погружения в процесс закрепления изменений в производственных процессах. Таким образом, регулярный мониторинг состояния производственного процесса в восприятии инженеров отягощается дополнительным аудитом со стороны менеджера проекта. Более логичным представляется разделение ответственности за контроль исполнения новых производственных практик между руководителем и лидером команды проекта.

В соответствии с личным опытом и опытом своих компаний в управлении изменениями эксперты оценили результаты производственных преобразований (рис. 12).

Участники исследования высказали свое мнение о зависимости между числом попыток внедрения изменений и их влиянием на конечные результаты (рис. 13). Это мнение подчеркивает, что в восприятии инженеров:

- спиралевидный подход к внедрению изменений более логичен и востребован;
- при стабильности коллектива инженеры каждую следующую попытку изменений в процессе производства программного продукта оценивают более благоприятно.

Анализ результатов внедрения изменений в производство ПО очень важен, однако не менее важно организовать его своевременно. Эксперты со ссылкой на опыт своих компаний отметили, что в основном анализ результатов внедрения изменений проводится по времени значительно позже их завершения (рис. 14).

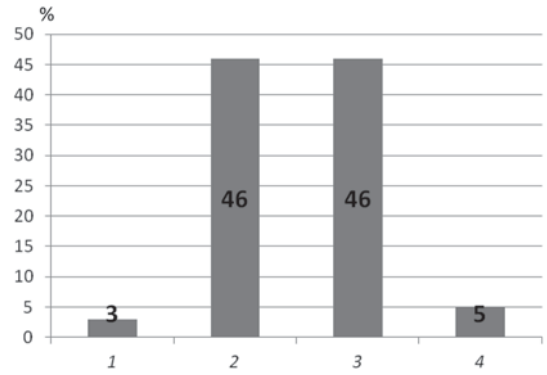


Рис. 12. Насколько успешно обычно достигаются цели внедрения изменений в производстве ПО:

1 — почти все цели утрачиваются; *2* — часть целей утрачивается, детали меняются; *3* — достигается большая часть целей; *4* — цели достигаются, а результаты превосходят ожидания

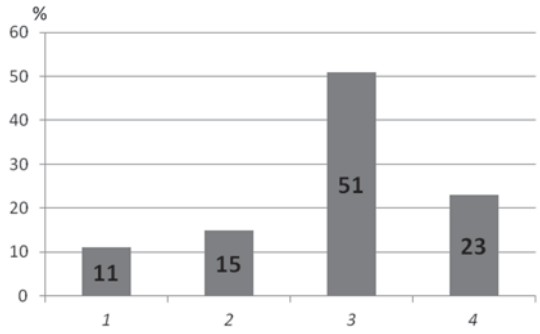


Рис. 13. В рамках одного коллектива является ли потенциально более успешной каждая следующая попытка внедрения изменений в производстве ПО:

1 — каждая следующая попытка имеет меньше шансов, чем предыдущая; *2* — количество попыток внедрения изменений не имеет значения; *3* — каждая следующая попытка имеет больше шансов, чем предыдущая; *4* — затрудняюсь ответить

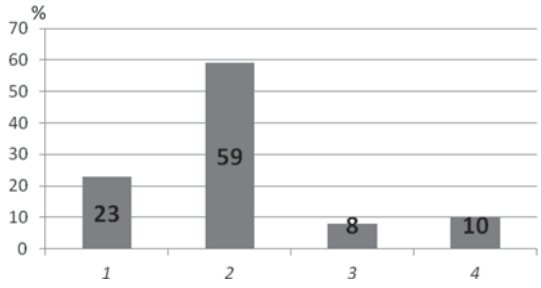


Рис. 14. Когда проводится анализ завершенного внедрения изменений в производстве ПО:

1 — сроки могут быть любыми; *2* — анализ проводится спустя несколько месяцев после внедрения; *3* —анализ проводится перед планированием следующих изменений; *4* — никто не проводит никакого анализа

Внедрения изменений в производственную деятельность должны быть удобными для сотрудников, которые являются основным нематериальным активом IT-компании. Отметим, что негативное влияние изменений на текущие производственные показатели должно быть минимизировано. Эксперты согласовали свое видение возможной степени регулярности внедрения значительных изменений в производственную модель (рис. 15), при этом они отметили, что в рамках одного проекта (или итерации проекта) внедрения двух значительных изменений в производственные процессы следует избегать. Из комментариев экспертов следует, что подразумевались проекты (итерации проектов) длительностью 4...6 месяцев.

Данный результат в совокупности с ожиданиями успехов от идущих подряд попыток преобразований и сроков подведения итогов может стать некоторой основой расписания регулярного внедрения изменений в производство, которое осуществляется централизовано. Суммируя все временные промежутки на планирование, внедрение и подведение итогов можно примерно оценить минимально комфортную для инженеров периодичность внедрения значительных изменений в производство, которая составляет 7...8 месяцев. Синхронизация таких итераций с производственными итерациями в проектной деятельности является вопросом, требующим дополнительного исследования.

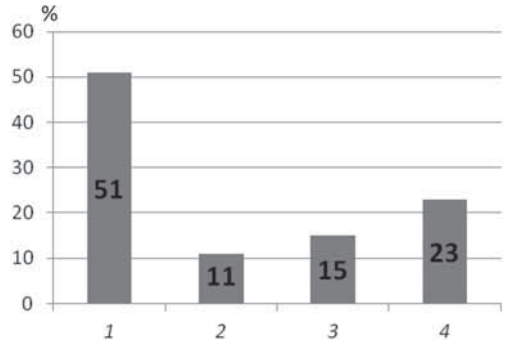


Рис. 15. Какой период времени считается комфортным между двумя идущими подряд внедрениями значительных изменений в процессы разработки ПО:

1 — в рамках одного проекта этого следует избегать; *2* — несколько месяцев; *3* — несколько недель; *4* — затрудняюсь ответить

Заключение

Исследование, результаты которого представлены в настоящей статье, обобщает опыт разработчиков ПО и их восприятие изменений в производственных процессах, а также подходы к их внедрению в практику. В исследовании рассмотрены все стадии внедрения таких изменений — от планирования до закрепления в производственную деятельность. Значительное внимание уделено рискам, отражающим специфику создания программного продукта, и возникающим при этом трудностям и способам их преодоления.

В целом эксперты считают, что стандартизация производственных процессов оказывает прямое влияние на качество программных продуктов. При этом в восприятии инженеров необходимым и естественным является раннее информирование разработчиков о будущих изменениях в производственных процессах, а также прозрачный характер планируемых изменений с самых ранних этапов выполнения проектов.

На этапе планирования и подготовки коллектива к изменениям эксперты выделили ряд основополагающих ожиданий и поделились своим опытом. Так, в восприятии инженеров централизованные изменения зачастую не являются таковыми: инициатива изменений, как правило, размыта среди различных руководителей компании (линейных и проектных менеджеров, руководителей офиса управления проектами, директоров направлений). Экспертное сообщество оценило также значительную часть изменений в производстве, как возникающие спонтанно. Этот факт свидетельствует о плохо построенных процессах раннего информирования и вовлечения членов проектных команд в процесс планирования изменений. Эксперты отметили также особую роль менеджера проекта, как одну из основных в процессах инициирования и управления производственными изменениями. Это обстоятельство указывает на то, что в этой роли сходятся несколько коммуникационных линий, в том числе и линия, ориентированная на совершенствование производственного процесса, которая положительно воспринимается инженерным составом.

Рекомендации, относящиеся к этапу планирования и подготовки коллектива к изменениям, могут быть сформулированы следующим образом: необходимо формализовать действия, направленные на совершенствование производственных процессов со стороны заинтересованных

в этом структур (Дирекция качества, SEPG, Офисы процессного развития и т. п.), компании следует обеспечить широкое информирование членов коллектива разработчиков о данной деятельности, поддерживая прямой контакт таких структур с инженерами компании. Руководителям проектов без должной необходимости не следует быть инициаторами в области совершенствования процессов разработки ПО, так как эта деятельность отнимает слишком много сил и времени от их основных управленческих задач. Исключение, пожалуй, составляет гибкое и экстремальное программирование, где изменения (в том числе в области производственных процессов) являются частью роли руководителя проекта [10]. Вместе с тем проектные менеджеры должны быть вовлечены в изменения, поддерживать их и, согласно ожиданиям экспертов, выявленным в исследовании, нести частичную ответственность за успешный результат использования этих изменений на практике.

Участвующие в исследовании эксперты представили свой опыт в области подготовки коллективов к предстоящим изменениям. Он заключается в том, что в большинстве случаев сотрудникам не предоставляется достаточно комфортного промежутка времени на подготовку. Выделенные экспертами приемы подготовки коллективов к изменениям являются довольно стандартными, а именно — информирование его членов на различных уровнях и вовлечение в разработку изменений. При этом эксперты указали, что существует некоторый буферный период, в течение которого изменения уже запланированы, но не начинают внедряться.

Как рекомендацию следует выделить то обстоятельство, что определяемые параметры буферного срока подготовки команд разработчиков и сроки информирования коллектива о предстоящих изменениях в совокупности с ожиданиями инженеров по обеспечению прозрачности подготовки изменений означают следующее:

- начало планирования изменений инициативной командой практически совпадает с началом раннего информирования коллектива о предстоящих изменениях;
- планирование изменений должно учитывать буферный срок подготовки, зависящий от совокупности конкретных условий в IT-компании.

Данные положения должны найти свое отражение в плане внедрения изменений, синхронизируя различные этапы работ по внедрению изменений в производстве.

Эксперты выделили ряд наиболее распространенных причин изменений в производственных процессах. При этом в восприятии инженеров экономически обоснованная объективная необходимость является наиболее распространенным случаем. Этот факт означает, что аргументы экономического характера обязательно должны быть доведены до сведения вовлеченных в производственный процесс сотрудников.

Непосредственное внедрение изменений, по мнению сообщества экспертов, может и должно быть поддержано на различных уровнях:

- на проектном уровне, контролируемом в том числе руководителем проекта;
- на корпоративном уровне — в виде формализованных приказов и бизнес-процессов;
- с помощью автоматизации производственных операций, которая поддерживает изменения.

Эксперты отметили следующие проблемные вопросы, которые типичны на этапе внедрения изменений:

- формальное внедрение изменений без результатов и понимания их целей;
- конфликты между целями проекта и целями внедрения изменений;
- резкое падение качества ПО и отставание по срокам поставки релизов.

Набор рекомендаций, направленных на преодоление отмеченных вопросов и рисков неблагоприятных событий, довольно стандартен. К ним относятся: раннее информирование сотрудников об изменениях с акцентом на экономические выгоды для проекта; сопряжение целей проекта и целей внедрения изменений в производстве; учет возможного падения качества создаваемого продукта и отставания по срокам реализации проекта, которые должны быть предусмотрены в карте рисков. Очевидно, что последняя рекомендация в карте рисков сопровождается набором корректирующих воздействий, которые включают дополнительные временные запасы, приоритизацию требований, дополнительные итерации тестирования.

Эксперты также выделили издержки, сопровождающие процесс внедрения изменений, к которым относятся:

- издержки по качеству и/или срокам поставки продукта;
- ухудшение "внутреннего климата" в команде, выполняющий проект;
- уход части специалистов из компании/проекта.

В восприятии экспертов организационное сопротивление является существенным препятствием в управлении изменениями. При этом набор организационных мер для его преодоления и мотивации сотрудников к принятию изменений довольно традиционен: разъяснительная работа; положительная мотивация; вовлечение в изменения и т. п. Интересно, что негативная мотивация, связанная с общественным порицанием или денежными штрафами, не воспринимается инженерами как эффективный подход к закреплению изменений.

Сообщество экспертов определило набор мер и мероприятий по закреплению изменений. Центральную роль среди них занимают усилия руководителя проекта, что косвенно подтверждает "слабость" специализированных структур типа SEPG или Офиса процессного развития и непрозрачность их деятельности при закреплении изменений в производственной деятельности. Безусловно, менеджер проекта может быть вовлечен в аудит состояния дел на этом направлении, однако выполнение такого аудита на постоянной основе несомненно отвлечет его от решения основных управленческих задач и приведет к издержкам.

Несмотря на довольно позитивное восприятие результатов внедрения изменений в производственную деятельность, инженеры все-таки находят ряд существенных замечаний в части организации централизованных изменений в производстве. Во-первых, спиральный метод с набором итераций представляется наиболее перспективным и оправданным, так как последовательность изменений в совокупности делает процесс постоянного улучшения производства ПО более структурированным и предсказуемым. Во-вторых, частота изменений должна быть синхронизирована с производственными (проектными) итерациями таким образом, чтобы не перегружать команды разработки задачами в области управления изменениями. По результатам исследования можно сделать предположение, что поддерживаемые менеджментом и централизованные итерации по улучшению производственных процессов должны проводиться не чаще, чем один раз в 7...8 месяцев.

Следует также отметить, что инженеры в недостаточной степени вовлечены в управление производственны-

ми изменениями и связывают основную активность на этом направлении (как, впрочем, и ответственность за результат) с руководителями проектов. Деятельность формальных структур IT-компаний (таких как SEPG, Офис процессного развития, Дирекция качества) для инженеров носит неясный и не совсем открытый характер. При этом в восприятии экспертов общее централизованное управление изменениями должно быть более прозрачным для команд разработки, а внедрение изменений должно проводиться формализовано, структурировано и плавно. Представленные результаты подчеркивают необходимость более явного и активного вовлечения сотрудников проектных команд в управление производственными изменениями.

Список литературы

1. **Ансофф И.** Стратегическое управление. М.: Экономика. 1989, 519 с.
2. **Harrington H. J.** Change Management Excellence: The Art of Excelling in Change Management, book 3. The Five Pillars of Organizational Excellence. Paton Press, 2007. 176 p.
3. **Камерон Э., Грин М.** Управление изменениями. М.: Добрая книга, 2006, 360 с.
4. **Published** CMMI Appraisal Results, URL: https://sas.cmmiinstitute.com/pars/
5. **Rowe G., Wright G.** The Delphi technique as a forecasting tool: issues and analysis // International Journal of Forecasting. 1999. Vol. 15, Issue 4. P. 353—375.
6. **Пащенко Д. С.** Проблемы внедрения изменений в проекте разработки и внедрения программного обеспечения // Известия ТулГУ. Экономические и юридические науки. 2014. Вып. 4. Ч. I. С. 303—314.
7. **Паулк М., Куртис Б., Хриссис М. Б.** Модель зрелости процессов разработки программного обеспечения. М.: Интерфейс-Пресс, 2002. 256 с.
8. **Боровко Р.** Российские компании-разработчики ПО улучшают свои процессы, используя CMM/CMMI. URL: http://www.cnews.ru/reviews/free/offshore/cmm/ (дата обращения 01.01.2015).
9. **Королев В. А., Стариков Н. П.** Основы системно-процессной теории устройства и жизнедеятельности организаций // Менеджмент и менеджер. 2007. № 11—12. С. 14—25.
10. **Дак Дж. Д.** Монстр перемен: Причины успеха и провала организационных преобразований. М.: Альпина Паблишер, 2007. 320 с.
11. **Pollice G.** Leadership in an (almost) Agile world. The Rational Edge, 2009. URL: http://www.ibm.com/developerworks/rational/library/edge/09/mar09/pollice/index.html

D. S. Pashchenko, MBA, Consultant in Software Development, e-mail: denpas@rambler.ru, Moscow

Research in CEE-Region: Changes Implementation in Software Production

Delphi method research have been done in middle of 2014, and 78 software engineers from 11 countries of CEE-region took part in it. In this article there are main results of this research, including: main problems and risks of change management, expert's experience in change implementation, aspect of production process model standardization in software business, engineer's emotional perception of changes. Expert's panel defined organizational resistance features in IT domain, need of timely changes implementation planning and consolidation, aspects of increasing the predictability impact of changes on software project's results. Also there are detailed recommendations to reduce the impact of the risk on the successful implementation of changes on all stages of internal project.

Keywords: *production changes, software development, organizational resistance in software domain, changes consolidation, staff involvement in change management*

References

1. **Ansoff I.** *Strategicheskoe upravlenie* (Strategic management). Per. c angl. Moscow: Jekonomika, 1989, 519 p. (in Russian).
2. **Harrington H. J.** *Change Management Excellence: The Art of Excelling in Change Management*, book 3. *The Five Pillars of Organizational Excellence*, Paton Press, 2007, 176 p.
3. **Cameron E., Green M.** *Making Sense of Change Management: A Complete Guide to the Models Tools and Techniques of Organizational Change*, Kogan Page, 2004, 280 p.
4. **Published** CMMI Appraisal Results, available at: https://sas.cmmiinstitute.com/pars/
5. **Rowe G., Wright G.** The Delphi technique as a forecasting tool: issues and analysis. *International Journal of Forecasting*, 1999, vol. 15, issue 4, pp. 353—375.
6. **Pashchenko D.S.** Problemy vnedrenija izmenenij v proekte razrabotki i vnedrenija programmnogo obespechenija (Issues of changes implementation in project of software development). *Izvestija TulGU. Jekonomicheskie i juridicheskie nauki,* 2014, vol. 4, part I, pp. 303—314 (in Russian).
7. **Paulk M. C., Curtis B., Chrissis M. B., Weber C. V.** Capability Maturity Model for Software, Version 1.1, Software Engineering Institute, CMU/SEI-93-TR-24, February 1993.
8. **Borovko R.** Rossijskie kompanii-razrabotchiki PO uluchshajut svoi processy, ispol'zuja CMM/CMMI, available at: http://www.cnews.ru/reviews/free/offshore/cmm/ (in Russian).
9. **Korolev V. A., Starikov N. P.** Osnovy sistemno-processnoj teorii ustrojstva i zhiznedejatel'nosti organizacij. *Menedzhment i menedzher,* 2007, no. 11—12, pp. 14—25 (in Russian).
10. **Dak Dzh. D.** *Monstr peremen: Prichiny uspeha i provala organizacionnyh preobrazovanij* (The Change Monster). Moscow: Al'pina Pablisher, 2007, 320 p. (in Russian).
11. **Pollice G.** Leadership in an (almost) Agile world. *The Rational Edge*, available at: http://www.ibm.com/developerworks/rational/library/edge/09/mar09/pollice/index.html

ООО "Издательство "Новые технологии". 107076, Москва, Стромынский пер., 4
Технический редактор *Е. М. Патрушева.* Корректор *Е. В. Комиссарова*

Сдано в набор 05.05.2015 г. Подписано в печать 20.06.2015 г. Формат 60×88 1/8. Заказ PI715
Цена свободная.

Оригинал-макет ООО "Адвансед солюшнз". Отпечатано в ООО "Адвансед солюшнз".
119071, г. Москва, Ленинский пр-т, д.19, стр.1. Сайт: **www.aov.ru**