УДК 004.4

Разработка систем автоматизированной оценки заданий по программированию

Иртегов Д.В. (Новосибирский государственный университет),

Нестеренко Т.В. (Институт систем информатики СО РАН, Новосибирский государственный университет),

Чурина Т.Г. (Институт систем информатики СО РАН, Новосибирский государственный университет)

В статье рассмотрена двадцатилетняя история и опыт создания автоматизированных систем проверки знаний и навыков по программированию в Новосибирском государственном университете. Приведен анализ эффективности или неэффективности методов и средств в использующихся системах проверки знаний. Описано современное состояние архитектуры системы NSUts.

Ключевые слова: *тестирование знаний, обучение программированию, олимпиады по программированию, система тестирования, NSUts, защита от мошенничества.*

1. Введение

Для оценки качества образования активно используется метод автоматизированного тестирования знаний и навыков. В различных формах этот метод применяется в школьном, высшем и послевузовском обучении. Сертификаты многих компаний, например, Microsoft Certified System Engineer, Cisco Engineer, Certified Lotus Professional выдаются на основе автоматизированного тестирования.

Эффективным комплексным средством проверки знаний и навыков программиста является написание программы, соответствующей заданным требованиям, с последующим её тестированием. Такие системы отличаются от традиционных автоматизированных систем тестирования тем, что они предполагают написание испытуемыми программ, последующий запуск которых осуществляется на заранее подготовленном наборе тестов.

В настоящее время опубликовано немного статей по автоматизированной проверке знаний, тем не менее, в разных странах такие системы существуют, и они достаточно популярны и востребованы, особенно при проведении различных соревнований по

программированию, так как проверять масштабные соревнования методом ручного тестирования затратно. Но, как правило, при работе таких систем необходимо присутствие автора.

Одной из первых, созданных в России, систем автоматизированной проверки была система ***T/Run*** для MS-DOS. Программа ***run*** обеспечивала запуск решений с ограничением по времени работы, а программа ***t*** использовала ***run*** для проверки решения на заданных тестах.

Система ***olympiads.ru*** [10] в своей основе содержит программу запуска решений с ограничением времени работы и использования памяти. Она дополнена базой данных задач и простым графическим интерфейсом пользователя, обеспечивающим редактирование базы данных и запуск решений на проверку.

На Всероссийских олимпиадах использовалась система ***Cyber Judge,*** созданная М.А. Бабенко. Но использование этой системы без автора затруднительно, так как настройка предполагает редактирование файлов конфигурации, кроме того, часто настройка системы предполагает дописывание к ней небольших модулей для конкретной олимпиады.

В качестве своеобразной системы самотестирования можно отметить систему ***tchoose***, используемую на Всероссийской олимпиаде школьников по информатике во время составления тестов для визуальной оценки их прохождения заранее заготовленными правильными и неправильными решениями.

Первой попыткой создать интегрированную систему стала связка Automated Programming Problem Evaluation System (APPES) на Java и Programming Contest Management System (PCMS) на Delphi, созданные в Санкт-Петербургском государственном институте точной механики и оптики. Их развитие превратилось в проект ***PCMS2*** [11]. Благодаря низкой связности модулей и качественному выделению высокоуровневых концепций достигается огромная гибкость. Однако при настраивании системы ***PCMS2*** большинство действий в нем осуществляется написанием файлов конфигурации, которые составляют значительную часть системы, как по важности, так и по объему, связывая ее модули вместе. В настоящий момент сервер ***PCMS2*** поддерживает только ОС Windows.

Олимпиады московского и некоторых других регионов проходили под управлением системы ***ejudge***, созданной в Московском государственном университете. Она является в некотором смысле противоположностью ***PCMS2***. ***Ejudge*** поддерживает только операционную систему Linux, написана на языке Си и поддерживает заранее определенный набор функций. Недостатком системы является возможная конкуренция за ресурсы между

тестируемым решением и самой системой. Тестирование проходит на одной машине, и подключить дополнительные физически невозможно.

Система автоматической проверки с архивом задач в испанском городе Вальядолиде [2] достаточна популярна среди студенческого сообщества. При работе используется операционная система Linux и языки программирования: GNU C, GNU C++, Free Pascal. Архив содержит сотни задач с тестами и постоянно пополняется. Для многих неудобным является использование системы Linux. Главным же недостатком данной системы и систем ***timus*** [1] Уральского государственного университета и ***Yandex Contest*** компании Яндекс [12] является то, что они используют модель Software as Service (SaS).

Для понимания функциональности и архитектуры системы NSUts, созданной в Новосибирском государственном университете (НГУ), в данной статье рассмотрена история разработки этой системы, описаны ее предшественники: системы на языке REXX, Chjudge, Uhjudge, аргументированы выбранные решения и проведен анализ их эффективности.

Опыт разработки и эксплуатации ряда разных программных решений сыграл важную роль как при определении требований к системе NSUts, так и при формировании архитектуры.

2. «Система» на REXX

Работы по автоматизации соревнований по программированию велись в Новосибирском государственном университете с 1998 года. Первое программное решение представляло собой скрипт на языке REXX длиной около тысячи строк. Оно имело многочисленные неустранимые недостатки и никогда не рассматривалось, как что-то пригодное для постоянной эксплуатации.

Однако, этот скрипт использовался более трех лет для проведения тренировок и соревнований, в том числе, для четвертьфинальных соревнований ACM-ICPC (Association for Computing Machinery – International Collegiate Programming Contest [7]), проводившихся в Новосибирском государственном техническом университете в 1999 году, а также Открытой Всесибирской Олимпиады по программированию в 2000 и 2001 годах [15, 16].

Историю этого решения следует начать с ноября 1997 года, когда был проведен первый региональный полуфинал чемпионата ACM-ICPC по Северо-Восточному Европейскому региону. НГУ получил официальное приглашение участвовать в этом мероприятии. Результаты команд НГУ трудно было назвать блестящими [13].

Руководством НГУ и заинтересованными преподавателями было осознано, что без систематической работы достичь приличных результатов в соревнованиях такого рода

невозможно. Первоначально обсуждалась идея наладить автоматическое тестирование при помощи скриптов командной оболочки MS/DR DOS *command.com* (*.bat*-файлов), но быстро было осознано, что возможностей языка *command.com* для решения этой задачи недостаточно.

Осенью 1998 года от организаторов Всероссийской олимпиады школьников была получена система, которая называлась PCMS, и была непосредственным предшественником ***PCMS2***. Эту систему планировалось использовать для проведения университетской олимпиады, по результатам которой должны были быть отобраны команды для участия в полуфинале ACM-ICPC.

В архиве файлов этой системы было обнаружено несколько бинарных исполняемых файлов для MS DOS, несколько конфигурационных файлов и никаких сопроводительных текстов. Было ясно, что формат конфигурационных файлов очень сложен, и по нему невозможно понять, в какой среде должна была исполняться система. В архиве был найден исполняемый модуль, функции которого удалось понять и которым можно было управлять при помощи параметров командной строки. Это была утилита ***run.exe***, которая позволяла запускать программы для DOS с контролем времени исполнения. Максимальное время задавалось параметром командной строки, при истечении этого времени запущенная программа принудительно завершалась. Также, утилита ***run*** отлавливала некоторые варианты нештатного завершения запущенной программы, например, ошибку среды исполнения Turbo Pascal. По результатам исполнения, утилита выдавала один из трех возможных вердиктов:

- «Успешное исполнение» (программа штатно завершилась с кодом возврата 0),
- «Ошибка времени исполнения» (программа штатно завершилась с кодом возврата, отличным от 0, или завершилась из-за ошибки среды исполнения)
- «Превышен лимит времени» (программа была принудительно завершена утилитой).

Д.В. Иртегов написал на языке REXX скрипт, использующий эту утилиту, который совмещал в себе реализацию очереди решений, тестовой обвязки и подсчета рейтинга.

Архитектура «системы» в целом выглядела следующим образом. Скрипт запускался на выделенном компьютере под Windows 95. Выбор ОС был продиктован следующими причинами:

1. Тестировать решения планировалось под DOS на языках программирования Borland C и Turbo Pascal. DOS-эмуляторы OS/2 и Windows NT имели многочисленные проблемы и ограничения, так что в них сложно было добиться корректной работы компиляторов этих языков. Возможно также, утилита ***run.exe*** не смогла бы корректно работать в этих эмуляторах. Это оставляло только две возможности: использовать настоящую DOS или DOS-окно Windows 95.

2. Интерпретатора REXX для DOS не существовало, был доступен только REXX/Win32, работающий под Windows NT и Windows 95. Выбор языка сложно было назвать осознанным решением. Фактически, использовался такой продукт, для которого имелся интерпретатор, и который был пригоден для решения задачи.

3. Операционная система Windows 95 обеспечивала несколько более качественную изоляцию программ для DOS, чем чистая DOS. Некоторые ошибки времени исполнения, которые под чистой DOS приводили к зависанию компьютера и *холодной* перезагрузке, под Windows 95 приводили к снятию задачи без перезагрузки всей «системы». Впрочем, полноценной изоляции Windows 95 не предоставляла, так что нередко все-таки приходилось делать холодную перезагрузку.

В качестве очереди решений использовалось дерево каталогов файловой системы. Это дерево должно было размещаться на сетевом файловом сервере. В нашем случае использовался сервер Novell Netware. В принципе, можно было использовать любой файловый сервер, который обеспечивал доступ со стороны клиентов DOS/Windows 95 и управление доступом на уровне пользователей. Последнему требованию на 1998 год удовлетворяли системы OS/2 LAN Manager и Windows NT, а позднее появился сервер Samba для ОС семейства Unix. Для проведения интернет-тура Открытой Всесибирской Олимпиады участники получали доступ к своим каталогам по протоколу FTP.

Компьютер с «системой» должен быть подключен к серверу от имени пользователя, имеющего доступ ко всему дереву каталогов. Для каждого участника соревнований создавалась отдельная учетная запись, которая имела доступ только к одному каталогу в дереве. Таким образом, участники не могли видеть решения друг друга.

Чтобы отправить решение на тестирование, участник должен был положить файл с исходным текстом решения в свой каталог дерева. Имя файла должно было соответствовать номеру задачи, а расширение — языку, на котором написана программа.

«Система» сканировала все каталоги дерева по очереди. Обнаружив файл с решением, она выполняла следующие действия:

1. Перемещала его в рабочий каталог и копировала в архивный каталог, добавляя к имени файла префикс, соответствующий имени каталога участника.

2. Определяла язык программирования и номер задачи.

3. Запускала соответствующий компилятор.

4. Проверяла результат компиляции и, в случае неудачи, выдавала вердикт «ошибка компиляции».

5. При успешной компиляции, последовательно запускала полученную программу под контролем утилиты **`run.exe`** на наборах заранее подготовленных тестов. Наборы тестов хранились в отдельных каталогах с именами, соответствующими номерам задач. Каждый набор представлял собой наборы файлов с именами 1.in, 1.out, 2.in, 2.out и т. д. Легко понять, что имя файла соответствовало номеру теста, а расширение — типу файла: входные данные или эталонные (правильные) выходные данные.

6. Если ***`run.exe`*** выдавала вердикт «Успешное исполнение», «система» сравнивала выходной файл программы с эталонным файлом правильного ответа при помощи утилиты ***`diff`***. Использовалась версия ***`diff`*** из поставки Borland C. Позднее была реализована также более сложная схема проверки при помощи программы, предоставленной жюри.

7. Если программа в процессе работы «подвешивала» компьютер и приходилось его перезагружать, то это обнаруживалось следующим образом: при нормальном запуске «системы» рабочий каталог должен был быть пуст. Если там лежали какие-то файлы, предполагалось, что они остались от неудачной попытки запуска. Это интерпретировалось как ошибка времени исполнения.

По результатам выполнения всех предыдущих шагов, «системы» могла получить один из следующих вердиктов:

- «Ошибка компиляции» (программа не скомпилировалась).
- «Ошибка исполнения на тесте N» или «Лимит времени на тесте N». Это определялось по соответствующим вердиктам утилиты ***`run`***, а также по наличию файлов в рабочем каталоге после перезагрузки.
- «Неверный ответ на тесте N». Этот ответ определялся по результатам сравнения выдачи программы с эталоном.

- «Принято». В этом случае программа участника выдала ответы, соответствующие эталонным, на всех тестах.

Определив вердикт, «система» выкладывала его в каталог участника в виде файла с именем, соответствующим номеру задачи, и с расширением .RES. Кроме того, «система» вносила вердикт в общий файл рейтинга.

Файл рейтинга представлял собой текстовый файл, в котором каждая строка соответствовала результатам одного участника. Строка была разбита на позиции при помощи символа `'|'`. В позициях, соответствующих номерам задач, стояло текущее состояние, закодированное следующим образом:

1. пробел соответствовал задаче, которую участник сдавать не пытался.
2. Знак '−' и десятичное число соответствовали одной или нескольким неудачным попыткам сдачи задачи.
3. Знак '+', возможно с последующим десятичным числом, соответствовал успешно сданной задаче. Число при этом соответствовало количеству неудачных попыток.
4. В последней позиции хранилось штрафное время, вычисляемое по правилам соревнований ACM. При этом, временем сдачи задачи считалось время модификации файла с исходным текстом.

Таким образом, файл был похож по формату на таблицу рейтинга соревнований ACM-ICPC. Для хранения рейтинга нужно было использовать какую-то базу данных, но в тот момент этого еще не было реализовано. Позднее, в 2001 году, Д.В. Иртегов этот недостаток устранил, переписав скрипт на Perl, вместо текстового файла была использована хранимая хэш-таблица Berkeley DB.

Среди наиболее важных недостатков и проблем этой «системы» следует упомянуть следующие.

1. Нетранзакционную процедуру отправки решения на проверку. Участник обнаруживает, что задача передана на проверку, по исчезновению файла из каталога. Это неудобно для пользователя и порождает целый класс возможных ошибок соревнования (*race condition*). Особенно чувствительным это неудобство было при работе через протокол FTP.

2. Невозможность отслеживать очередь тестирования для участников.

3. Нетранзакционную работу с базой данных рейтинга. Файл с рейтингом считывался и разбирался заново после каждого тестирования каждого решения, он перезаписывался полностью после каждого изменения рейтинга. Сбой сети, сервера или рабочей станции в

этот момент мог привести к потере данных. Кроме того, при пересчете рейтинга «система» делала много лишней работы. С переходом на Berkeley DB этот недостаток был, в основном, устранен.

4. Хранение только последнего отправленного участником решения. Это делало невозможной корректную обработку некоторых апелляций. Так, если в предоставленных жюри тестах будет обнаружена ошибка, то, следует проверить все решения всех участников на откорректированном наборе тестов, и зачесть как успешную ту попытку сдачи, которая первая пройдет все тесты. Но, поскольку «система» хранила только последнее решение, это оказывалось невозможным. Значимость этого недостатка была осознана после того, как такая апелляция была принята и корректно обработана системой на финале ACM-ICPC 2000.

5. Невозможность горизонтального масштабирования. Поскольку выбор решений из каталогов нетранзакционен, то запустить несколько экземпляров «системы» на разных машинах для параллельного тестирования невозможно. Работу с файлом рейтинга можно было бы синхронизовать, используя механизм блокировок файлов, но из-за неразрешимой проблемы с выборкой решений не делалось даже попыток реализовать это.

6. «Система» выбирает решения не в соответствии с временем их отправки, а в соответствии с лексикографическим порядком имен каталогов, так что решения участников с лексикографически неудачными именами тестируются позднее.

7. «Система» не защищена от простого мошенничества, когда участник задает произвольное время модификации файла с решением. Технически это несложно, в DOS есть соответствующий системный вызов, и это приведет к существенным нарушениям в алгоритме подсчета рейтинга по правилам ACM. К счастью, за все время эксплуатации системы никому из участников это не пришло в голову.

8. «Система» не предоставляла удобных средств коммуникации между участниками и жюри. Правила олимпиад ACM, на которые мы ориентировались, предусматривали такую коммуникацию в форме вопросов участников и ответов жюри. В нашей «системе» такая функциональность была реализована следующим образом. Если участник загружал в свой каталог файл с расширением .QUE (Question), то «система» вместо его компиляции выдавала на консоль тестового компьютера сообщение «Вопрос от участника». Член жюри, сидящий за другим компьютером, мог получить доступ к файлу с вопросом и положить в каталог участника другой текстовый файл с расширением .ANS (Answer). Это работало, но было очень неудобно и для участников и для жюри.

Как уже говорилось, все эти недостатки были ясны уже при разработке решения. Одной из главных причин, по которой решение со всеми этими проблемами все-таки эксплуатировалось, было мнение, что разрабатывать «нормальную» систему для тестирования задач под DOS не имеет смысла. Штатная среда исполнения — чистый DOS или Windows 95 — не обеспечивает полноценной изоляции, поэтому плохо себя ведущая программа может «подвешивать» ОС. Следовательно, компьютер, на котором проводится тестирование, должен находиться под постоянным присмотром. Таким образом, следует говорить не об автоматизированном, а о механизированном тестировании, и неполная адекватность средств такой механизации не является главной проблемой.

3. Система CHjudge

В 2000 году произошло несколько событий, которые заставили отказаться от ранее описанной «системы» и заняться разработкой полноценного решения.

Во-первых, команда НГУ вышла в финал ACM-ICPC, и на финале произошел эпизод с перетестированием, который произвел на нас большое впечатление. Стало ясно, что если даже на таких высокоранговых соревнованиях случаются ошибки в тестах, то и наши университетские мероприятия от этого не застрахованы. Следовательно, процедура тестирования должна допускать справедливую по отношению к участникам обработку таких ситуаций.

Во-вторых, соревнования Северо-Восточного Европейского полуфинала ACM-ICPC, наконец-то, отказались от DOS и перешли на тестирование заданий под Windows. NT обеспечивает гораздо более качественную изоляцию пользовательских процессов, так что «плохо себя ведущая» программа практически не может нарушить функционирование системы в целом. Таким образом, главный аргумент, который удерживал нас от разработки полноценной системы, потерял силу.

В-третьих, организация интернет-тура Открытой Всесибирской Олимпиады по программированию вызвала нарекания у многих участников, и значительная часть этих нареканий была связана с системой тестирования.

В обсуждении архитектуры новой системы принимали участие наши финалисты ACM-ICPC: Евгений Четвертаков, Александр Шапеев и Алексей Бабурин, которые на тот момент были студентами НГУ, Д.В. Иртегов, Т.Г. Чурина, сотрудник ИСИ СО РАН С.К. Черноножкин. Довольно быстро был достигнут консенсус по следующим требованиям к системе:

1. Должно поддерживаться проведение соревнований по правилам ACM-ICPC с тестированием программ на языках C, C++, Pascal для Win32.

2. Язык программирования должен указываться участником явно при отправке задания, а не определяться по косвенным признакам, таким, как расширение файла. Это позволило бы проводить в рамках одного тура тестирование программ на нескольких диалектах одного языка, например, Visual C и Borland C.

3. Система должна предоставлять участникам веб-интерфейс для отправки задач, просмотра состояния их тестирования и рейтинга, а также для коммуникации с жюри.

4. Сами задачи, очередь тестирования и рейтинг должны храниться в транзакционном хранилище, скорее всего в реляционной СУБД.

5. Должна сохраняться полная история всех попыток отправки решения, все исходные тексты и точные времена их отправки. Это необходимо как для обработки апелляций, требующих перетестирование, так и для расследования попыток мошенничества.

6. Тестирование решений должно производиться на выделенных компьютерах. Главной мотивацией этого требования было то, что время исполнения задачи в многозадачной среде определяется сложными сценариями конкуренции за разные ресурсы: оперативную память, кэш процессора, время переключения контекста процессора и т. д. Исполнение решения на выделенном компьютере позволило бы свести влияние такой конкуренции до минимума.

Кроме того, исполнение задач на выделенных компьютерах позволило использовать разные операционные системы. На тестирующих компьютерах («тестовых клиентах») могла использоваться Windows NT, а для веб-сервера системы и сервера базы данных — Linux. Это оказалось особенно важно потому, что в 2000 году не было приемлемых по производительности и надежности веб-серверов для Windows NT.

Пункт 6 привел к разделению системы на два компонента:

- центральный сервер, предоставляющий веб-интерфейс, а также хранилища тестов, очереди решений, рейтинга и другой информации;
- тестирующие клиенты.

За разработку системы взялся Евгений Четвертаков, к тому моменту уже имевший некоторый опыт разработки веб-приложений. В качестве языка разработки был выбран язык Perl, в то время довольно популярный для реализации серверной части веб-приложений. В качестве сервера СУБД был использован MySQL.

Серверная часть системы представляла собой типичное веб-приложение, общающееся с пользователями – участниками и членами жюри, при помощи HTML-форм и хранящее персистентные данные в СУБД.

Логика работы тестирующего клиента была похожа на логику работы скрипта «системы», рассмотренной в предыдущем разделе. Разница состояла в том, что клиент содержал логику работы с очередью решений и тестовую обвязку, а логика вычисления рейтинга была перенесена на сервер. Несмотря на сходство логики, код скрипта «системы» в коде клиента не переиспользовался. Большая часть кода клиента была реализована на Perl, часть – в виде .BAT файлов для CMD.EXE (командного процессора Win32).

Отдельным вопросом при разработке архитектуры системы был механизм взаимодействия центрального сервера с тестирующими клиентами. Логически возможны два варианта:

1. Режим проталкивания (*push*), когда клиент ожидает запроса от сервера на тестирование задачи. При появлении задачи в очереди, сервер выбирает одного из клиентов и передает ему задачу.

2. Режим вытягивания (*pull*), когда клиенты в холостом цикле опрашивают сервер, не появилось ли у него новой задачи для тестирования. Защита от передачи задачи двум клиентам одновременно осуществляется за счет использования транзакционной очереди, реализованной на основе СУБД.

На первый взгляд кажется, что вариант проталкивания более привлекателен. Действительно, при вытягивании, клиенты должны постоянно опрашивать сервер, что создает паразитную нагрузку на него. Однако при реализации системы был принят режим вытягивания, не столько из-за простоты его реализации, сколько из-за соображения, которое можно кратко сформулировать следующим образом: при малых нагрузках режим проталкивания бесполезен, а при больших он не работает.

Дело в том, что тестирование каждого отдельного решения – это длительный процесс, состоящий из многих шагов, каждый из которых может завершиться неудачей: ошибка компиляции, ошибка на первом тесте, ошибка на втором тесте и т.д. Предсказать время тестирования каждой отдельной программы невозможно. В худшем случае, с точки зрения нагрузки, программа может пройти все тесты, затратив на каждый максимально допустимое время, а в лучшем случае она может быть снята по ошибке компиляции.

Поэтому, передав решение клиенту, сервер не может определить, когда этот клиент освободится, и даже не может оценить интервал, с которым этот клиент будет обновлять информацию о своем состоянии.

Большую нагрузку на систему можно определить, как состояние, когда значительную часть времени почти все клиенты заняты тестированием. В этих условиях, сервер не имеет свободных клиентов и не может определить, когда они освободятся. Поэтому режим проталкивания оказывается неработоспособен.

Паразитная нагрузка на сервер, создаваемая опросом со стороны вытягивающих клиентов, возникает только в условиях, когда клиенты ничем не заняты, то есть в условиях низкой нагрузки. В этой ситуации, паразитная нагрузка не представляет самостоятельной проблемы.

Разработанное Е.А. Четвертаковым программное решение оказалось весьма удачным. Оно удовлетворяло всем осознаваемым на тот момент требованиям, обеспечивало высокую надежность и приемлемую производительность даже под довольно высокими нагрузками.

В 2003 году система была использована для проведения интернет-тура Открытой Всесибирской Олимпиады. В интернет-туре олимпиады участвовало 102 команды.

До 2007-2008 года, система Четвертакова активно использовалась в НГУ для проведения Всесибирской олимпиады по программированию, тренировок команд НГУ и ряда других мероприятий. Были попытки ее использования в учебном процессе в курсе «программирование на языке высокого уровня» на ФИТ и ММФ НГУ.

Первоначально система поддерживала проведение соревнований только по правилам ACM-ICPC, и соответствующая бизнес-логика организации тестирования и правила подсчета рейтинга были жестко закодированы. Позднее был добавлен модуль, поддерживающий проведение соревнований по правилам российской школьной олимпиады по информатике. В этом модуле и правила, и бизнес-логика были также жестко закодированы.

Наиболее важным отличием с точки зрения бизнес-логики является следующее:

- При тестировании задачи по правилам ACM-ICPC, принятая задача обязана успешно пройти все тесты, т.е. получить вердикт «принято». Таким образом, если на каком-то тесте получается вердикт, отличный от «принято», последующие тесты можно не запускать.
- По правилам российской школьной олимпиады, задача получает определенное количество баллов за каждый принятый тест, и не обязана проходить все тесты, чтобы

эти баллы были включены в рейтинг. Поэтому, если программа выдала «неверный ответ» или «ошибку времени исполнения», тестирование необходимо продолжать.

Оказалось, что система вполне работоспособна без участия разработчика, благодаря достаточно хорошо написанным пошаговым инструкциям, как развернуть систему и подготовить ее к проведению мероприятия.

В ходе эксплуатации был выявлен ряд недостатков, как функциональных, так и нефункциональных.

Среди наиболее важных функциональных недостатков, требовавших пересмотра архитектуры системы, следует назвать следующие.

1. Система была рассчитана на однократное проведение одного мероприятия. Для проведения другого мероприятия с другими задачами и тестами необходимо было развернуть систему заново, создав новую базу данных. Конфигурация системы была рассчитана на сосуществование нескольких экземпляров системы на одном сервере, поэтому данная операция не требовала уничтожения данных предыдущих мероприятий. Для проведения соревнований этот недостаток не казался важным, но для тренировок и использования в учебном процессе это превратилось в существенную проблему.

2. Система обеспечивала веб-интерфейс для большинства сценариев штатного функционирования, но многие нештатные ситуации, например, дисквалификация участника соревнований или удаление ошибочно созданной учетной записи, требовали прямой модификации базы данных.

3. Система имела очень простое управление привилегиями с разделением всех учетных записей на две категории: участников и жюри/администраторов. При проведении крупных мероприятий с многочисленным жюри и техническим комитетом это превращалось в существенную проблему. Так, доброволец-студент, работа которого состояла в том, чтобы распечатывать исходные тексты программ по запросам участников и разносить их по терминальным классам, должен был иметь административный доступ к системе, что давало ему технические возможности останавливать и продолжать тур, отвечать на вопросы от имени жюри и т. д. Случаев злоупотребления этими правами зафиксировано не было, но само существование такой возможности вызывало у оргкомитета серьезные опасения.

Кроме названных функциональных недостатков, система имела и ряд существенных архитектурных и реализационных проблем, а именно:

1. Система была реализована с грубыми нарушениями принципа разделения содержания, представления и поведения. Каждая страница веб-интерфейса генерировалась отдельным скриптом, который выглядел, скорее, как HTML-код со вставками кусочков кода на Perl, чем как программа. В начале 2000-х годов такой стиль веб-программирования рекомендовался многими учебниками и размещенными в интернете руководствами, но уже к середине десятилетия он был признан плохой практикой. Это затрудняло как изменения дизайна веб-интерфейса, к 2007-2008 году интерфейс уже выглядел устаревшим, так и доработку функциональности и бизнес-логики системы.

2. Код системы не имел выделенного слоя модели данных. SQL-запросы к БД были включены непосредственно в код скриптов, генерирующих веб-страницы. Изменение модели данных могло потребовать внесения согласованных изменений во все файлы исходного кода приложения. Это также признано в отрасли плохой практикой и сильно затрудняло доработку и поддержку системы.

3. Тестирующие клиенты опрашивали очередь решений при помощи прямых запросов к СУБД. Это снижало безопасность, например, приходилось хранить в коде системы имя и пароль для доступа к СУБД, и затрудняло решение некоторых вспомогательных задач. Практически невозможно было отслеживать состояние клиента через веб-интерфейс системы. Если по какой-то причине клиент зависал или вовсе отключался, система не могла это обнаружить.

4. К 2007 году версия Debian Linux, под которую была разработана CHjudge, была официально снята с поддержки. В более новых версиях Debian, некоторые библиотеки, использовавшиеся при реализации системы, были объявлены устаревшими и исключены из репозиториев Debian, а некоторые из них — и из репозиториев CPAN (Compherensive Perl Archive Network).

Устаревшие архитектура и стиль кодирования системы затрудняли привлечение студентов к ее доработке, а потребности в доработке нарастали. Кроме перечисленных выше крупных функциональных недостатков, накапливалось большое количество мелких функциональных и эргономических замечаний. Поэтому в 2007 году был поставлен вопрос о разработке новой системы с нуля.

4. Ujudge

Относительно успешный опыт разработки и эксплуатации Chjudge породил у оргкомитета Всесибирской Олимпиады и лично у авторов иллюзию, что задача автоматизации

соревнований по программированию – это задача студенческого уровня. Это утверждение может быть более подробно изложено следующим образом.

1. Функциональность серверной части системы тестирования достаточно проста и почти не содержит вычислительно сложных алгоритмов, поэтому серверный компонент не может оказаться узким местом в производительности системы в целом.

2. Вследствие предыдущего пункта, не нужно уделять большого внимания производительности серверной части. Можно сосредоточиться на сборе и формулировке точных функциональных требований и пожеланий к системе, и на точном воплощении этих требований в программном коде.

3. Выбор языка и платформы для разработки не играет большой роли. Действительно, Perl, на котором была реализована система CHJudge, представляет собой интерпретируемый язык со слабой динамической типизацией и поздним связыванием. Производительность таких языков обычно довольно низка, и по большинству синтетических тестов Perl находился примерно на одном уровне с другими популярными языками этого типа, такими, как PHP, Python или популярный в 2007-2008 годах Ruby.

4. Поскольку на пути от функциональных требований к законченному продукту нет существенных подводных камней, с разработкой по заданным требованиям может справиться любой достаточно мотивированный успевающий студент, специализирующийся в области информационных технологий, или небольшая группа таких студентов. Эти студенты, как и при разработке CHjudge, могут работать без плотного контроля со стороны опытных наставников. Также, поскольку мотивация студентов-разработчиков является одним из приоритетных параметров, студентам можно предоставить выбор языка и платформы для разработки.

Практика показала ошибочность всех этих пунктов, но именно ими мы и руководствовались при организации работ над новой версией системы.

Руководство разработкой новой версии системы взяла на себя преподаватель ФИТ и ММФ НГУ, участник жюри Всесибирской Олимпиады Т.Г. Чурина. Она же координировала сбор и формализацию требований к системе. Д.В. Иртегов выступал в роли консультанта и внес достаточно большой вклад, чтобы нести ответственность за результаты.

Главным источником требований к новой системе были те функции, которые успешно исполняла система CHJudge, а также списки замечаний к этой системе. Замечания можно

рассматривать как функции, которые были нужны жюри или участникам, и которые система не исполняла. Эти требования впоследствии легли в основу требований к системе NSUts.

Непосредственно разработкой занялся студент бакалавриата ФИТ А.В. Таранцов. По его предложению, разработка велась на языке Ruby с использованием фреймворка Rails. Несколько позже к группе разработчиков присоединились студенты ФФ А. В. Киров и С.Б. Факторович.

Весной 2008 года разработка новой версии была сочтена завершенной. А.В. Таранцов успешно защитил квалификационную работу бакалавра.

Архитектура новой системы, получившей название Ujudge, сохранила определенную преемственность с CHJudge. Система состояла из сервера, реализующего веб-интерфейс для работы жюри и участников соревнований, и тестирующих клиентов, на которых выполнялось, собственно, тестирование. Код и логика работы тестирующего клиента не подверглись сколько-нибудь значительным изменениям.

Серверная часть была полностью переписана с учетом новых требований на новом языке и новых технологиях. Модель данных также была спроектирована с нуля. Единственное, что у серверной части Ujudge было общего с CHJudge — это использование MySQL в качестве СУБД.

Наиболее заметным для пользователей усовершенствованием был пользовательский интерфейс системы. Он был полностью перепроектирован, введены элементы технологии AJAX. В частности, для получения оповещений об ответах на вопросы и новостей не надо было перезагружать веб-страницу.

Еще одной новой особенностью Ujudge был механизм подсчета рейтинга олимпиады. Заложенную в этом механизме идею можно описать как компромисс между

- «жестким кодированием», когда весь алгоритм подсчета рейтинга реализован как модуль на том же языке программирования, что и сама система. Этот модуль может без ограничений использовать все внутренние интерфейсы системы, поэтому для разработки новых модулей такого же типа необходимо глубокое знание внутренней организации системы, а реорганизация системы, в свою очередь, может потребовать переделки модуля.
- «мягким кодированием», когда алгоритм подсчета рейтинга описан в конфигурационном файле. Это требует введения в язык конфигурации конструкций, аналогичных языкам программирования (переменных, присваиваний,

условных операторов, циклов), а в пределе и вовсе разработки нестандартного полного по Тьюрингу императивного или декларативного языка. Это приводит к тому, что конфигурация системы становится очень сложной. Действительно, для стандартных языков программирования существует хорошая документация и разного рода инструментальные средства, облегчающие разработку: редакторы с синтаксическим подчеркиванием, отладчики, интегрированные среды и т.д. Нестандартный язык программирования ничего этого не имеет. Поэтому «мягкое кодирование» может сделать процесс настройки системы существенно более сложным, чем модификация исходного кода.

Идея компромисса состоит в том, чтобы предоставить пользователям системы API для разработки плагинов – модулей, взаимодействующих с остальной системой через выделенные хорошо документированные интерфейсы. Таким образом, пользователь может разработать свой модуль подсчета рейтинга по своим правилам, используя стандартный язык программирования и стабильный API.

А.С. Таранцов разработал сложную компонентную архитектуру, фактически целый фреймворк, для подсчета рейтингов по разным алгоритмам. Этот фреймворк включал в себя алгоритм построения таблицы путем вызова предоставленных пользователем, в данном случае – разработчиком рейтинга, функций. Для доступа к результатам проверки решений предоставлялись специальные классы языка Ruby, которые следовало использовать вместо прямых обращений к СУБД.

Система была опробована на тренировках сборной НГУ и получила положительные отзывы, хотя некоторые из пользователей жаловались на низкую наблюдаемую производительность. Но попытка ее использования для проведения очного-тура Всесибирской Олимпиады по программированию в сентябре 2008 года закончилась неудачей.

В очном туре 2008 года участвовало 46 команд. Во время пробного тура участники ознакомились с новым интерфейсом системы, и многие высказали положительные отзывы. В первые полчаса тура система вела себя вполне удовлетворительно, но потом стало заметно, что загрузка процессоров сервера растет. Приблизительно ко второму часу соревнований, загрузка всех процессорных ядер дошла до 100%, а через некоторое время сервер перестал реагировать на HTTP запросы. Сложилась сложная ситуация, но удалось быстро написать скрипт миграции данных и учетных записей из Ujudge в CHJudge и продолжить соревнования на «четвертаковской» системе.

Очный тур 2008 года проводился на CHjudge, но сам эпизод надолго запомнился всем участникам событий.

Анализ причин случившегося показал, что фреймворк подсчета рейтинга использовал алгоритм, вычислительная сложность которого и количество обращений к базе данных квадратично зависели от количества сданных задач. Пока сданных задач было мало, рейтинг вычислялся быстро. Когда количество сданных задач увеличилось, подсчет рейтинга не только занял все процессоры, но и заблокировал все остальные процессы через блокировки СУБД. Поскольку на тренировках количество сданных задач не достигало порога, при котором это происходит, тестирование на тренировках эту проблему не выявило.

В ходе последующего «разбора полетов» был достигнуто соглашение по следующим позициям:

1. Задача разработки полноценной системы автоматизации проведения олимпиад – это задача не студенческого уровня. Хотя привлекать студентов к разработке системы можно и, с педагогической точки зрения, даже нужно, разработка должна вестись под управлением и контролем со стороны взрослых.

2. В процесс разработки системы следует ввести полноценный, хотя, может быть, и упрощенный по сравнению с коммерческой разработкой, контроль качества, включающий отслеживание не только функциональных, но и нефункциональных требований, в первую очередь – производительности и отчуждаемости кода.

3. Разрабатывать новую версию системы с нуля не следует. Необходимо сосредоточиться на доработке существующего работоспособного кода. Нефункциональные требования по модифицируемости и отчуждаемости кода следует обеспечивать за счет поэтапного рефакторинга.

После принятия этих решений, встал вопрос, какую из имеющихся систем, Ujudge или CHJudge, следует принять за «существующий работоспособный код».

Анализ ситуации показал, что платформа Ruby on Rails, строго говоря, не является платформой. Разработчики Ruby и Rails используют политику разработки, известную как «катящееся обновление» (rolling update). Попросту говоря, все изменения в систему вносятся только в последнюю, «текущую» версию. Так, если в коде интерпретатора или библиотек обнаруживается ошибка, то она исправляется только в последней версии. Поэтому все разработчики, использующие Ruby On Rails, вынуждены постоянно синхронизовать среду разработки с основным репозиторием. Устаревших, но поддерживаемых версий,

аналогичных «стабильным» версиям продуктов с открытыми исходными текстами или релизам коммерческих программных продуктов, в Ruby не существует. Фактически, вся платформа представляет собой непрерывную бета-версию.

При этом, разработчики Ruby не очень заботятся о совместимости своей платформы с самой собой. Обновление на новую версию среды исполнения часто ломает существующие программы и требует внесения изменений в их исходный код.

Зафиксировать версию платформы невозможно, так как в ней могут быть обнаружены серьезные ошибки, например, уязвимости с точки зрения безопасности. Эти ошибки будут исправлены только в текущей версии системы, то есть придется обновлять всю среду, а это, в свою очередь, потребует внесения изменения в ваше приложение для устранения интерфейсов, которые разработчики Ruby сочли «устаревшими».

Таким образом, разработка и поддержка приложения на платформе Ruby on Rails также превращается в «катящееся обновление», темп которого задается разработчиками платформы. Для короткоживущих программных проектов, например, для учебных проектов или прототипов это может быть терпимо, но для продуктов с многолетним жизненным циклом абсолютно неприемлемо.

В современном Ruby политика разработки более адекватна требованиям продуктов с длительными жизненными циклами, но здесь описано положение дел, имевшееся в 2008-2009 годах.

Поэтому за базу для дальнейшей разработки была принята система CHjudge. Дальнейшая история этой системы – это уже история системы NSUts.

5. Система NSUts

Одним из основных требований, предъявляемых работодателями к выпускнику ВУЗа, является умение работать в команде. Наилучшим способом обучения навыкам командной работы является участие в реальном программном проекте, особенно если этот проект завершается успехом. Реальный проект, по определению, должен быть нацелен на достижение определенного, нужного заказчику, результата в определенные сроки. Однако подключение студентов к таким проектам сталкивается с рядом проблем, главной из которых является то, что студенты не имеют опыта работы, и, следовательно, проект с их участием с высокой вероятностью может завершиться срывом сроков, превышением бюджета или полной неудачей. Теоретически, эти недостатки можно было бы компенсировать созданием смешанных команд из «взрослых» разработчиков и студентов. Но найти достаточное

количество опытных разработчиков, согласных работать в высоко-рисковом низкобюджетном проекте, практически невозможно. Вопрос о методологиях командной разработки, которые могли бы снизить риски, создаваемые неопытными разработчиками, и, таким образом, повысить вероятность успеха проекта, не является целью этой статьи.

Требования, предъявляемые к автоматизированным системам тестирования знаний, а именно: обеспечение глубокого и адекватного тестирования знаний и навыков, эффективной защиты от мошенничества и простота в эксплуатации для специалистов средней квалификации раскрыты в статье [14]. Безопасность в системе NSUts изложена в статье [3], описание прототипа системы NSUts – в статье [4].

Автоматизированная система NSUts [9] состоит из трех основных подсистем: клиентского ПО, сервера олимпиад (учебных туров) и тестирующего клиента. Клиентское ПО (браузер и среда разработки, в которой пользователь разрабатывает и отлаживает свое решение) установлено на компьютере пользователя и не предоставляется системой, но, как легко понять, принимает активное участие в работе. Тестирующих клиентов в системе может быть несколько. Единственный выделенный узел системы – это сервер.

Рис. 1. Архитектура системы NSUts

Сервер тестирования реализует логику проведения олимпиады (тура) и выполняет следующие функции:

1. Предоставляет веб-интерфейс для взаимодействия участников и членов жюри (преподавателя) с системой тестирования.
2. Автоматизирует управление олимпиадой (туром), а именно, осуществляет:

- сбор и хранение условий задач, тестов и решений;
- обработку и отображение результатов тестирования;
- составление рейтинга мест, которые заняли участники соревнования;
- получение отчётов о проведении олимпиады (тура);
- перетестирование решений участников;
- решение задач организационного плана, таких как регистрация участников, обеспечение обратной связи с жюри и других;
- администрирование олимпиад и туров.

Тестирующий клиент непосредственно осуществляет прогон решения на тестовых данных. Решения участников, полученные сервером, помещаются в очередь решений, откуда они забираются тестирующим клиентом. Тестирующий клиент в процессе обработки решения получает исходный код решения, информацию о необходимом для компиляции решения компиляторе, набор тестов. Исходный код компилируется и запускается в изолирующей среде на наборе тестов, результат работы программы сравнивается с эталонными результатами. Результат тестирования отправляется обратно на сервер, где происходит его обработка, составление рейтингов и так далее. Участники олимпиад и члены жюри взаимодействуют с системой исключительно через веб-интерфейс и не взаимодействуют с тестирующим клиентом напрямую.

Сервер написан на языке Perl с использованием модулей архива CPAN и некоторых утилит командной оболочки (shell для Linux или cmd.exe для Windows). Работа сервера олимпиад осуществляется под управлением веб-сервера apache2 [18]. Хранение данных приложения осуществляется в базе данных MySQL. Сервер тестирования может быть запущен как ОС Linux, так и в ОС Windows.

В настоящее время ведется покомпонентная переработка сервера на архитектуру AJAX, когда сервер отдает пользовательским машинам только данные в формате JSON, а их преобразование в красивый и удобный пользовательский интерфейс производится уже на клиенте, с использованием программ на JavaScript, исполняющихся в браузере. По данным наших измерений [5], это, в сочетании с переписыванием серверной части на PHP, может снизить нагрузку на сервер, например, расходы процессорного времени и оперативной памяти сервера на формирование страницы очереди, в разы или даже в десятки раз. Это может быть очень важно при больших нагрузках. В 2017 году была реализована и внедрена AJAX версия самой загруженной подсистемы сервера, страницы очереди.

Тестирующий клиент также написан на языке Perl с использованием командной оболочки MS Windows cmd.exe посредством использования bat-файлов. Отдельные компоненты

реализованы на языке C с использованием WinAPI. Существуют версии клиента для MS Windows, Linux и Solaris. В настоящее время, главный спрос со стороны организаторов соревнований предъявляется именно на соревнования под Windows, поэтому поддержка клиентов Linux и Solaris приостановлена.

Тестирующий клиент для Windows средствами Win32 API ограничивает вычислительные ресурсы (память, процессорное и астрономическое время) для тестируемой программы. Предпринималось несколько попыток ограничить также права доступа для этой программы, но от них пришлось отказаться из-за неприемлемых накладных расходов по времени.

Время работы решения на одном тесте ограничено несколькими секундами. В последние годы у большинства задач лимит времени на прохождение одного теста составляет 1-2 секунды и редко превышает 10 секунд. Время запуска модуля тестирования должно быть того же порядка величины или меньше, так как при проведении крупных соревнований по программированию тестирующий клиент должен проверить до 5000 решений участников олимпиады на 50-100 тестах — это примерно полмиллиона запусков.

Самое простое решение, доступное в Win32, для запуска процесса с ограниченными полномочиями — это использование системного вызова CreateProcessAsUser. Была разработана версия изолирующей среды, использующая этот механизм, но измерения показали, что работа этого системного вызова занимает 1-2 секунды, что было сочтено неприемлемым.

В 2016 году магистрантом ММФ А.Э. Кимом была разработана и внедрена версия изолирующей среды, работающая как отдельный постоянно запущенный процесс [6]. Это открывает возможность для повышения уровня изоляции: действительно, такой процесс можно запустить из-под учетной записи с ограниченными правами один раз, и накладные расходы на смену учетной записи не будут влиять на каждый запуск задачи. К сожалению, из-за нехватки времени разработчиков, на февраль 2018 года, версия тестирующего клиента, развернутая в системе, этой возможностью не пользуется.

Тестирующие клиенты взаимодействуют с сервером через протокол HTTP, поэтому они могут быть размещены как на той же физической машине, что и сервер, так и на выделенных машинах, физических или виртуальных. В НГУ для тестирования олимпиад используются тестирующие клиенты на выделенных физических машинах.

Отказ от виртуализации обусловлен тем, что виртуализация сопровождается сложно учитываемыми накладными расходами, которые, в свою очередь, могут приводить к перерасходу процессорного времени по сравнению с выделенной физической машиной. В наших условиях это может приводить к ложным вердиктам «превышение лимита

времени». Вопреки распространенному мифу, накладные расходы гипервизоров не ограничиваются производительностью ввода-вывода. Гипервизор участвует во всех операциях управления памятью в гостевой ОС, в частности, при запросах пользовательских процессов на выделение виртуальной памятью и при страничной подкачке. В зависимости от стиля работы программы с памятью, эти накладные расходы могут привести к увеличению астрономического времени работы и процессорного времени по счетчикам гостевой ОС на десятки процентов по сравнению с физической машиной [19].

По аналогичной причине разработчики вынуждены были также отказаться от запуска нескольких клиентов на одной физической машине. По данным измерений [8], сам факт, что на других процессорных ядрах многоядерного процессора что-то исполняется, может приводить к существенному замедлению работы программы, главным образом, из-за конкуренции процессорных ядер за доступ к ОЗУ. Это в примерно равной степени относится как к общему (астрономическому) времени работы программы, так и к процессорному времени, измеренному по счетчикам ОС или по аппаратным счетчикам процессора. При использовании гипертрединга, работа программы может замедляться в несколько раз. Поскольку характер и величина замедления решающим образом зависят не только от характера обращений тестируемой программы к памяти, но и от того, что именно работает на других ядрах, данный эффект никак невозможно контролировать или компенсировать.

Поэтому, как уже говорилось, в системе NSUts все официальные соревнования проходят тестирование на выделенных физических машинах. Поскольку необходимо поддерживать ферму тестирующих клиентов, достаточно мощную для проведения соревнований, эти же клиенты используются и для тренировок, в процессе которых ложные вердикты не так опасны.

6. Заключение

Опыт создания и использования автоматизированной системы проверки заданий по программированию NSUts и ее предшественников позволил выработать требования, предъявляемые к системам такого рода [4]. На их основе разработана архитектура системы, состоящей из трех основных частей: клиентского ПО, сервера олимпиад и тестирующего клиента. Система обеспечивает устойчивую работу под высокими нагрузками и обрабатывает подавляющее большинство нештатных ситуаций, возникающих во время мероприятий. Кроме того, опыт показал, что система достаточно легко адаптируется к изменяющимся условиям (поддержка новых языков программирования, изменения правил соревнований).

Система NSUts была создана для обеспечения проверки решений участников олимпиад по программированию, поэтому главной частью ее использования является проведение олимпиад по программированию всех уровней. Она используется при проведении Открытой Всесибирской олимпиады по программированию им. И.В. Поттосина [17], районных и региональных школьных олимпиад по программированию в Новосибирской области. Например, в первых этапах Всероссийской олимпиады школьников по информатике, проводимых в Новосибирской области с помощью системы NSUts, одновременно участвовало несколько сотен школьников, было проверено свыше тысячи решений. Некоторые студенческие олимпиады собирают около 1000 участников, при этом проверяется в онлайн-режиме более 10000 решений в течение одного тура.

Круглосуточная работа системы NSUts позволила организовать не только поддержку тренировок по программированию, а также и использование ее в учебном процессе для проведения практических занятий по программированию для студентов младших курсов.

Стремительное развитие техники и технологий ставит новые задачи и перед нашей системой, которые приходится решать ежедневно. Поэтому описанная в этой статье история создания системы NSUts еще не закончена.

Список литературы

1. Архив задач с автоматической системой проверки в Екатеринбурге [Электронный ресурс]. URL: http://acm.timus.ru
2. Архив задач с автоматической системой проверки в испанском городе Вальядолиде [Электронный ресурс]. URL: http://acm.uva.es
3. Боженкова Е.Н., Воронков А.Д., Иртегов Д.В., Конышева Е.Н., Черненок С.А., Чурина Т.Г. Модель разграничения прав доступа в системе автоматизированной проверки корректности программных приложений // Вестник НГУ Серия: Информационные технологии. - 2011. - Том 09, Выпуск № 4. - С. 79-85. - ISSN 1818-7900.
4. Боженкова Е.Н., Иртегов Д.В., Киров А.В., Нестеренко Т.В., Чурина Т.Г. Автоматизированная система тестирования NSUts: требования и разработка прототипа // Вестник НГУ Серия: Информационные технологии. - 2010. - Том 08, Выпуск № 4. - С. 46-53. - ISSN 1818-7900
5. Боженкова Е. Н., Иртегов Д. В., Колбин Я. С. Оптимизация производительности веб-интерфейса приложения NSUts средствами динамического HTML // Вестн. Новосиб. гос. ун-та. Серия: Информационные технологии. 2015. Т. 13, вып. 2. С. 13–21.
6. Ким А.Э., Разработка и реализация новой архитектуры тестирующего клиента системы NSUts, магистерская диссертация, ММФ НГУ, 2017, науч. рук. Иртегов Д.В., Чурина Т.Г.

7. Сайт международной олимпиады по программированию ACM-ICPC [Электронный ресурс]. URL: http://icpc.baylor.edu.
8. Свиридов В.С. Измерение и контроль потребления ресурсов программами на машинах с многоядерными процессорами, выпускная квалификационная работа бакалавра, ФИТ НГУ, 2012, науч. рук. Иртегов Д.В.
9. Система NSUts [Электронный ресурс]. URL: https://olympic.nsu.ru/nsuts-new/login.cgi
10. Система olympiads.ru [Электронный ресурс]. URL: http://www.olympiads.ru/
11. Система PCMS2 [Электронный ресурс]. URL: https://neerc.ifmo.ru/school/spb/municipal-participant.html
12. Система Яндекс Контест [Электронный ресурс]. URL: https://contest.yandex.ru/
13. Таблица результатов полуфинала ACM-ICPC 1998 года [Электронный ресурс]. URL: http://neerc.ifmo.ru/past/1997/standings.html.
14. Чурина Т.Г, Иртегов Д.В. Требования к автоматической системе тестирования знаний// Труды VI Международной конференция "Интеллектуальные технологии в образовании, экономике и управлении", декабрь 2009, Воронеж, с. 309-317.
15. I Открытая Всесибирская олимпиада по программированию им. И.В. Поттосина [Электронный ресурс]. URL: http://olympic.nsu.ru/old-site/widesiberia/archive/wso1/2000/index.shtml
16. II Открытая Всесибирская олимпиада по программированию им. И.В. Поттосина [Электронный ресурс]. URL http://olympic.nsu.ru/old-site/widesiberia/archive/wso2/2001/ruls1.shtml
17. XVIII Открытая Всесибирская олимпиада по программированию им. И.В. Поттосина [Электронный ресурс]. URL https://olympic.nsu.ru/widesiberia/2017/news
18. Apache HTTP Server Version 2.2 Documentation [Электронный ресурс] URL: http://httpd.apache.org/docs/2.2/ .
19. [http://ieeexplore.ieee.org/document/6670704/].