Квадрига Аполлона и микропроцессоры

Иосиф КАРШЕНБОЙМ
iosifk@narod.ru

Скульптура с мраморного рельефа квадриги Аполлона, бога Солнца. Рельеф, относящийся к IV—II вв. до н. э. был найден Шлиманом при раскопках Трои

Спасибо читателям за отклики

Итак, господа читатели, мы уже выяснили связь между известной картиной в Русском музее и мажоритаром. Тем, кто заинтересуется этим сюжетом, могу предложить ссылку[1]. Особенно хочется поблагодарить В. Черникова за прекрасное письмо — отзыв о моих «Записках Инженера».

А теперь — новый сюжет. Причем строго в соответствии с рекомендациями ВНИИГПЭ. Итак, начинаем описание интересующего нас сюжета с краткого обзора его аналога.

Квадрига Аполлона

Квадрига Аполлона — это четверка коней, скачущая по небу. Четыре красавца коня и бог, который ими управляет. А если посмотреть прагматично, то это просто четыре двигателя, установленные для параллельной работы и идеально (божественно) управляемые. Данный принцип исправно действовал и действует, начиная с древних колесниц и до сегодняшних «Прогрессов» и «Протонов». Правда, цена двигательной установки растет, достигая четырехкратного увеличения. Ну так Аполлон и есть Аполлон, какая уж тут экономия? «А зачем вообще надо наращивать мощность двигательной установки?» — спросит читатель. Ответ очевиден: чтобы ездить быстрее. Да, вероятно уже древние «юзеры» знали: хочешь увеличивать мощность, ставь более мощный двигатель. А когда достигнут предел в единичной мощности — добавляй двигатели. Они также знали, что наращивание количества двигателей не ведет к арифметическому увеличению производительности.

Мало того, при подобном увеличении значительно усложняется управление всей двигательной установкой. И так — от Фаэтона до ракеты Н-1. Знали все древние «юзеры» и в назидание потомкам слагали легенды о том, что нарушение синхронизации работы двигательной установки ведет к катастрофическим отказам.

Вот об этом и пойдет рассказ дальше, о том, как «быстрее и мощнее».

Представляю читателям несколько статей под общим заголовком «Квадрига Аполлона». (Надеюсь, что получится несколько статей.) В этих материалах хочется обратить внимание читателей на то, что происходит при увеличении числа процессоров в проекте. И каким образом это увеличение сказывается на различных аспектах проекта.

Ну вот, теперь можно переходить к микропроцессорам.

Время, деньги и микропроцессоры

Почему вдруг об этом?

Когда автор собирал материал о сетевых процессорах, о коммутаторах третьего уровня, о многопоточности в микроконтроллерах, то совершенно неожиданно на глаза попался материал иного плана. Да, здесь тоже идет речь о том, что бывает при увеличении числа процессоров. Только в несколько другом ракурсе. А поскольку подобных материалов в отечественной печати крайне мало, то тема показалась автору довольно интересной. Но об этом судить вам, уважаемые читатели. Поэтому для начала давайте обсудим самое новое и интересное. Обещали «прочтение гарантируем», ну так за дело!

Несколько строк о том, «как»…

До сих пор я писал только о том «что», то есть целью статей было показать саму разработку, конструкцию, программу… Но вот теперь хочется затронуть тему, «как» делать разработку…

В «староглиняные времена» нас, молодых советских специалистов, учили в основном тому «что». Произвести расчет узла, выбрать комплектующие, исследовать осциллограмму. Скромный курс планирования производства был точно выдержан в советском стиле. Статичное, бескризисное и всем понятное. Даже то, с чем пришлось познакомиться в советских НИИ, подразумевало, что планы и сроки, которые назначались в ЦК и Совмине, там же и корректировались. Надо больше приборов — занесем в план поставки, и через пару лет… Надо больше людей — ответ примерно такой же.

В новые времена, там, где мне пришлось работать, вопросы «как» решались по-разному, где по наитию, а где и так, что иногда приходилось просто теряться в догадках на тему «зачем и почему босс делает именно так». Соответственно в области «как» накопилось много вопросов.

Но при изучении состояния дел в разработках многомикропроцессорных устройств и сетевых микропроцессоров мне довелось познакомиться со статьями Джека Ганссла (Jack G. Ganssle).

Джек Гансл был владельцем небольшой фирмы, производившей отладочное оборудование — внутрисхемные эмуляторы, необходимые для отладки программ в микроконтроллерах. Посещая своих клиентов, он получил большой опыт общения с разработчиками из многих американских фирм. Проводил семинары по встроенным системам и в качестве консультанта помогал компаниям справиться с их «встроенными» проблемами, возникающими при разработке встроенного программного обеспечения.

Этот опыт Гансл описал в статьях и книгах. На многих примерах рассказал о том, как надо разрабатывать изделия со встроенными микроконтроллерами. Но в данном случае хочется представить российскому читателю то, «как» ведут разработки американцы и что они делают для своевременного выполнения проектов. Поэтому, думается, надо попытаться описать процесс разработки программ с точки зрения Джека. Статьи Джека Ганссла могут быть найдены на его сайте — **ht tp:/ /ww w.ganssle.c om/**.

Наиболее интересной для рассмотрения представляется статья «Уменьшение стоимости программного обеспечения путем «добавления» процессоров» [2]. В ней рассмотрены способы уменьшения стоимости выполнения проекта. Возможно, отдельные моменты покажутся читателю спорными, тем не менее, автор данной статьи считает, что некоторые взгляды и суждения Джека Ганссла могут быть интересны не только отечественным разработчикам, но и их боссам.

И для начала хочется процитировать только один абзац из указанной статьи Джека Ганссла.

Ценность встроенного программного обеспечения стала дороже золота!

«Сегодня встроенное программное обеспечение — самая дорогая вещь во Вселенной. Председатель совета директоров фирмы «Локхид-Мартин» г-н Норман Огастин (Norman Augustine) в своей книге «Законы Огастина» (Augustine, Norman. Augustine's Laws, AIAA (American Institute of Aeronautics & Astronautics), Reston, VA, 1997) написал о тех интригах, которые были у производителей самолетов-истребителей 60-х годов. Чтобы увеличить прибыль, производители мечтали о том, чтобы добавить «что-нибудь» к конструкции самолета. Но уже к концу 70-х больше не было никаких возможностей добавить что-либо «физическое» к конструкции самолета, потому что добавление новых функциональных возможностей означало увеличение веса самолета, что в свою очередь привело бы к ухудшению всех его характеристик. Однако продавцы самолета все-таки мечтали добавить «нечто», чтобы увеличить свою прибыль. Они искали это «нечто», такое, которое бы ничего не весило, но позволило значительно увеличить стоимость. И они нашли это «нечто» — это было встроенное программное обеспечение! Это был ответ на их мечты.

Самолет F-4 был лучшим истребителем 60-х. Его стоимость в долларах, по уровню цен 2004 г., составляет приблизительно $20 млн за штуку. Истребитель F-4 не имел встроенного программного обеспечения. А сегодняшний, новейший серийный F-22 стоит уже $257 млн. И половина цены — это встроенное программное обеспечение.

Вот так жизнь обогнала даже самые смелые мечты производителей о росте прибыли».

Джек Ганссл

Далее речь пойдет о том, что необходимо сделать, чтобы избежать ошибок в коде программы и увеличить эффективность процесса проектирования. Все, что здесь описано, будет полезно также и для FPGA дизайнеров, поскольку технологии кодирования для языков высокого уровня примерно одинаковы.

Почему разделение на части настолько важно?

«Краткосрочная память человека, скорее всего, подобна крошечному кэшу, который фактически может держать в голове только пять–девять вещей прежде, чем новые данные заменят там предыдущие данные. Большие функции переполняют умственный кэш программиста. Программист больше не может полностью понять код, следовательно, ошибки в коде распространяются».

Джек Ганссл

Как ведется проект? Написать код и отладить. Снова написать и снова отладить. И так до тех пор, пока проект не будет завершен. Причем отлаживать проект приходится по двум причинам. Первая — это ошибки в коде, вторая — неправильное задание для проектирования. Вторая причина приводит, как правило, к более существенным переработкам проекта, чем первая. Но поскольку обе причины влияют на производительность труда, их можно оценивать интегрально, как ошибки в проекте. Глобально избежать появления ошибок невозможно. А как часто они появляются и насколько при этом снижается производительность труда?

Чтобы понять, откуда берутся ошибки, необходимо определить, насколько комфортно чувствует себя человек на производстве. Вспомните кадры старой хроники, на которых были сняты советские КБ военного и послевоенного времени. Длинные ряды кульманов и рабочих столов, над которыми склонились конструкторы. Белые халаты, тишина и только голос диктора, рапортующий о победах. В эпоху «развитого» дело обстояло так. «Личный состав» отечественных КБ обычно набирался с тем запасом, чтобы всегда без большого ущерба для дела можно было отвлечь часть инженеров в колхозы, на овощебазы и пр. Поэтому помещений обычно не хватало. Столы теснились один к другому. Стулья с занозами… Но, скажет читатель, это ж когда было! Да, настали новые времена. Красивые офисы инофирм, блестящая мебель, чистые стены. А вот стало ли удобнее работать? Да, во многих фирмах теперь применяется гибкий график рабочего дня, так что те работники, которые имеют склонность поработать во второй половине дня, — могут приходить позже и соответственно позже уходить с работы. Но вот телефон за соседним столом как был противный и громкий — так и остался. Или обсуждение проблем любимой футбольной команды. Этим лучше всего заниматься в комнате отдыха, но как часто бывает, такие места вовсе не предусмотрены. Да, аренда площадей дело дорогое, но ведь разработчики еще дороже, хотя часто об этом и не задумываются. Ну а теперь немного статистики. Результаты исследований, проведенных Томом Демарко и Тимоти Листером [3] в течение 10 лет, показали следующее. Если испытуемую группу разбить на четыре части по убыванию производительности, то их результаты можно свести в таблице 1. В первой группе были собраны результаты самых производительных работников. В четвертой — наименее производительных. И вот что выяснилось: отвлечения и прерывания работы приводят к уменьшению производительности труда.

Таблица 1. Корреляция производительности с прерываниями

	1-я четверть	4-я четверть
Выделенное рабочее место	8 м²	4,5 м²
На рабочем месте тихо?	57% да	29% да
Оно принадлежит только вам?	62% да	19% да
Можете ли вы выключить телефон?	52% да	10% да
Можете вы не отвечать на вызовы?	76% да	19% да
Частые прерывания?	38% да	76% да

Что же интересного в этой таблице? Вот итог исследований: первая группа была на 260% более производительней, чем четвертая!

Теперь давайте еще раз вспомним, как в кинофильмах выглядят американские офисы. Помните — большой зал, но всегда разбитый на маленькие клетушки. Так, что для работника соседи не видны и их разговоры его не отвлекают.

И еще одно замечание. Те, у кого компьютер стоит дома и кому приходится работать по вечерам, хорошо знают такую ситуацию. «Давай скорей выключай, ужин подгорит!» И никакие слова, что компьютер не телевизор и его нельзя выключить вот этой кнопкой «Вкл.», здесь не помогают. Так то — компьютер, он выгрузит систему и снова загрузит ее, когда угодно и сколько угодно раз. Другое дело — человек. Был у автора этой статьи несколько лет назад момент, когда редактор текста Word сбивался несколько раз подряд и вылетал без сохранения. Помнится, первый раз текст был такой радостный и легкий, после первого вылета — текст был уже не радостный, а после третьего — напоминал некролог. Человеку необходимо время для того, чтобы для продуктивной работы произвести «загрузку» головы и хорошего настроения. И «загрузка» эта может достигать в среднем 15 мин. Если человека отвлекать хотя бы раз в 15 мин., то производительность такого работника окажется равной нулю. Посмотрите последние строки в таблице 1 — это и есть тот самый случай, когда отвлекают от работы.

Следовательно, необходимо, чтобы в наиболее продуктивное для данного работника время его не отвлекали от непосредственного выполнения задачи. Тривиально? Да! Но все же — отделиться перегородкой от соседа, соблюдать тишину, отключить телефоны, не устраивать совещания, ну и конечно, не обсуждать футбол… И, разумеется, иметь возможность где-то передохнуть и расслабиться. Когда один мой бывший коллега уволился из инофирмы и вернулся на государственное предприятие, где заработки были ниже, вот что он ответил на мой вопрос: «Почему, Дмитрий?» «А потому, что там негде было поговорить, и как только в курилке собиралось три человека, приходил «ответственный» и строго предупреждал о том, что больше трех собираться запрещено». Вот такой был ответ, хотя здравым умом подобную методу работы понять невозможно.

Теперь о том, что проект проекту рознь

Но самая главная проблема состоит в том, что разработчики, создающие большие системы и подсистемы, имеют намного меньшую производительность, чем те, кто пишет

крошечные программы. Странно, но вот какое высказывание можно здесь привести для обсуждения. Например: «Сто строчек кода я сейчас напишу за 10 минут (20 минут, 30 минут... в зависимости от «крутизны»)». Это высказывание отнюдь не вызывает удивления. Но тогда, если рассуждать в рамках простой арифметики, большие проекты в тысячи и сотни тысяч строк кода должны делаться за считаные дни и недели. Однако в жизни такого не происходит. Следовательно, при увеличении объема проекта продуктивность падает.

Сначала посмотрим, что происходит при простом увеличении проекта

В таблице 2 приведены данные, собранные из различных проектов по разработке программного обеспечения, выполнявшихся в фирме IBM [4]. Из таблицы видно, что производительность работы программиста резко уменьшается в соответствии с увеличением объема проекта, поскольку у более сложных и больших по объему проектов растут возможности и варианты выполнения! И, как говорит Джек Ганссл, «получается, что ситуация точно противоположна тому, чего так громко требует босс».

Таблица 2. Производительность работы программиста в месяц, выраженная в строках программы (по данным фирмы IBM)

Объем проекта в человеко-месяцах	Продуктивность в строчках кода программы за месяц
1	439
10	220
100	110
1000	55

А вот что происходит, когда над проектом работает несколько человек

Теперь надо несколько слов сказать о том, что проекты бывают разной «толщины». Помните фразу «А в попугаях-то я — больше!»? Вот теперь самое время применить такую же методику и к измерению проектов. Только мерить будем не в попугаях, а в головах разработчиков. Если весь проект «помещается» в одной голове, то он выполняется с производительностью, которую имеет владелец той самой головы. А если не «помещается»?

Фред Брукс в своей оригинальной работе «Мифические человеко-месяцы» [5] описал, как вырос проект 360/OS фирмы IBM, для выполнения которого первоначально требовалось 150 человек, а в результате он «дорос» более чем до 1000 разработчиков. Причем все неистово писали записки, делали сообщения и совершенно немного программного кода. В 1975 году был сформулирован закон Брукса, гласивший, что добавление людей к проекту, который уже отстает от графика, приводит только к тому, что это отставание становится еще больше. Таким образом, проекты по разработке программ, находящиеся на грани провала, по-прежнему подтверждают этот принцип. Что же делают управленцы? Они все еще продолжают бросать работников на критические проекты в ошибочной вере, что проект объемом в N «человеко-месяцев» может быть закончен через четыре недели N программистами.

Так в чем же дело? Почему падает производительность в больших проектах? Причина в том, что как только проект перестает «помещаться» в одной голове, так тут же в дело вступает «человеческий» фактор. Когда проект ведется одним человеком, он выполняет проект так, как он умеет, привык и на том языке и теми средствами, которыми обычно пользуется. И ему никому не надо объяснять что и как, и каким образом взаимодействует в его проекте. Даже если проект выполнен как «лоскутное» одеяло, и состоит из одних заплаток, имеет мало комментариев, он все же работает. Обычно с этим приходится мириться. А вот при работе в «команде» такой способ работы уже не проходит. И кроме файлов программ появляются многочисленные описания, протоколы согласования и пр. И чем больше проект, тем больше специалистов по «чудесам» делопроизводства участвуют в нем. Письма, протоколы согласования, графики выполнения работ, телефонные звонки, совещания, оперативки. А производительность? Да некогда теперь программы писать! Объем информации, необходимой для согласования усилий разработчиков, растет как снежный ком. Таблица 3 иллюстрирует результаты анализа, произведенного Джоулом Ароном в IBM [5]. Производительность работы программистов резко падает на больших системах, главным образом, из-за взаимодействий, необходимых для членов группы.

Таблица 3. Производительность резко упала при увеличении взаимодействий

Взаимодействия	Продуктивность LoC/man-year — строки кода на человека в год
Очень мало	10 000 LoC/man-year
Среднее	5000 LoC/man-year
Много	1500 LoC/man-year

Каков же выход из этой ситуации? Как представляется автору статьи, выход в том, что руководитель проекта должен очень тщательно прорабатывать документацию, необходимую для согласования и разработки проекта. В данном случае «прорабатывать» совсем не означает делать всю документацию самому. Скорее наоборот. Документацию должны разрабатывать те, кто находится в непосредственной близости к проекту — а именно разработчики. Задача же руководителя — проверить эту документацию и определить пути ее разработки так, чтобы всем исполнителям был ясен дальнейший план. Для примера приведу факт из отечественной истории. «Луна твердая!» — так написал Королев на докладной записке о том, какую стратегию необходимо применить при разработке спускаемого аппарата. И станция села на поверхность Луны и не утонула в лунной пыли. Таким образом, когда есть кардинальное направление развития, действия всех групп разработчиков скоординированы. Причем объем взаимного общения значительно сокращается, и у разработчиков появляется время для работы. Чтобы не быть голословным, приведу пример, описанный в [6]. В этом примере показано, что такое задание на проектирование и к чему приводят ошибки в составлении технического задания.

А что же происходит в худшем случае? Когда действия разработчиков не согласованы? Вот пример из проекта по разработке PCI-target в FPGA, который выполнял автор статьи много лет назад. Сделал проект, но для уверенности в правильности работы надо было проверить, как себя вел один бит в автомате. Прихожу к программисту, прошу вывести в окно параметр регистра, чтобы проверить значение этого бита. Ответ был такой: «Я системный программист, у меня серьезные дела, а вы тут с каким-то битом». Жду пару дней, наконец получаю доработанную программу. Смотрю, что и как работает. Работает, вот только в окно выводятся не те значения. Снова жду пару дней… Ну, и так далее. Снова приходилось идти к «крутому системному программисту» и просить исправить его программу. О какой производительности труда при таком подходе к делу можно говорить?

Эффект суперпрограммиста

Разработчиков было принято делить на три категории. МНС, НС, СНС и ВНС — это для научных сотрудников. Младший, средний (он же просто научный сотрудник), старший и ведущий. А для инженеров — 1, 2, 3 категории и еще «ведущий» и «главный». Больше делений и не было. При этом считалось, что есть некоторая «норма» выполнения должностных инструкций. Участвуешь в лыжных кроссах, в ДНД, в заседаниях парткома — значит… Ну и так далее. Однако пришли новые времена. Градаций стало еще меньше — «инженер» и «сеньор инженер». А вот, как говорится, «по жизни» разница между производительностью и качеством выполнения работы разных людей может отличаться на порядок. Но только здесь надо сделать небольшое отступление. Вспомните, когда у вас были периоды максимальной производительности? Наверное, так: вам дали небольшую часть проекта, вас вызвал к себе босс и сказал, что надо срочно сделать очень важную часть проекта, которую некому поручить, кроме вас, потому что это задание совсем новое и не связано с тем, что делают все остальные, и вся фирма смотрит на вас и т. д. И вот, на удивление, вы сделали свою часть легко и быстро. И работа была не такая нудная, как обычно, когда надо «стыковать» свою часть проекта с остальным проектом, ждать, когда другие части проекта

будут готовы и можно будет этот кусок отладить и т. д.

Данное наблюдение подтверждается исследованиями, проведенными Capers Jones [7]. Результаты этих исследований приведены в таблице 4. Как показано в том же исследовании, кривая, описывающая производительность разработчиков, имеет форму колокола. Плохие и хорошие разработчики — по краям. Средние, как и положено, в середине. Лучшие разработчики, суперпрограммисты, превосходят других на маленьких работах. Бросьте их на огромный проект, и они будут работать приблизительно с той же самой производительностью, как и самые слабые разработчики в группе.

Из этого можно сделать лишь один вывод: большие проекты стирают суперзвезд. Мораль ясна и критически важна: мудрые менеджеры дают своим самым лучшим разработчикам маленькие части проектов, выделенные из большого проекта. Разделите вашу систему на несколько процессоров и позвольте суперпрограммистам сделать отдельные части проекта, только тогда вы получите дополнительный выигрыш производительности от суперпрограммистов.

Таблица 4. Эффект суперпрограммиста

Размер в KLoC	Лучший программист (мес/KLoC)	Худший программист (мес/KLoC)
1	1	6
8	2,5	7
64	6,5	11
512	17,5	21
2048	30	32

И еще немного о суперпрограммистах. Часто в телеконференциях задают один и тот же вопрос: на каком языке лучше всего программировать? VHDL, Verilog или дискуссии о применении C во встроенном программном обеспечении? Ответ может быть только один, и этот ответ автор неоднократно повторял в подобных мероприятиях: программировать нужно на том языке, на котором работает лучший программист фирмы. Это, во-первых, самые лучшие «образцы для подражания», во-вторых — самые лучшие библиотеки программ. И, наконец, в-третьих, с программным обеспечением, она всегда сможет привлечь самую квалифицированную «скорую помощь».

Планирование работы

В настоящее время 70–80% стоимости разработки изделия определяется стоимостью создания программного кода. Вероятно, подобное отношение будет только увеличиваться, потому что по данным, приводимым в иностранной печати, объем встроенного программного обеспечения удваивается каждые 10 месяцев.

Что такое планирование работы программистов и зачем оно нужно? Почему такие работы столь тщательно велись на Западе? Ответ очень прост. Все дело в способе финансирования производства. Производство существует либо на собственные средства, либо на заемные. И у нас в стране большинство предприятий ведет работы за счет собственных средств. Если вы ведете производство на собственные средства, то небольшая задержка в поставке вашей продукции на рынок повлечет лишь некоторое снижение прибыли. И практически не скажется на цене поставляемого продукта. Совсем другое дело, когда приходится использовать заемные средства. Кстати, на Западе большинство предприятий ведет свою деятельность за счет заемных средств — это и ценные бумаги, и банковские кредиты. А неплатежи по кредитам в банк могут обернуться очень большими проблемами. И если вы пришли к инвестору и предложили ему сделать «нечто» под его кредит и просрочили платеж или поставку этого «нечто», то вполне возможно, что инвестор предоставит вам дополнительное финансирование, но одновременно пересмотрит цену того изделия, которое он хочет от вас получить. Так что при неумелом планировании вы можете потерять значительную долю прибыли, а то и всю прибыль. Но и это еще не самое плохое. Худший вариант — инвестор или банк откажут вам в дополнительном финансировании, и тогда ваша фирма — банкрот. Все то, что вы имели и разрабатывали, ваши сотрудники, на обучение которых вы потратили столько сил и средств, — все это уйдет с молотка. В лучшем случае вы получите работу как наемный сотрудник. Или придется искать работу самому. И знайте, что информация о вашем банкротстве будет учитываться всеми вашими новыми кредиторами, если таковые найдутся.

Вот именно с этой точки зрения и надо рассматривать то, что с появлением нового вида деятельности — программирования, начались работы по планированию, определению стоимости работы и прогнозированию сроков ее исполнения.

Продолжение следует

Литература

1. ht tp:/ /w ww .iosifk.nar od.r u/engineer_storys.ht m
2. Subtract software costs by adding CPUs. Jack Ganssle, Embedded.com.
3. Demarco, Tom and Timothy Lister. Peopleware : Productive Projects and Teams, Dorset House Publishing Company, New York, NY, 1999.
4. Jones, Capers. Applied Software Measurement: Assuring Productivity and Quality, McGraw Hill, New York, NY, 1991.
5. Brooks, Frederick. The Mythical Man-Month: Essays on Software Engineering, 20th Anniversary Edition, Addison-Wesley Professional, New York, NY, 1995.
6. ht tp:/ /iosifk.naro d.r u/nat_m3.pdf
7. Jones, Capers. Private study conducted for IBM, 1977.
8. Boehm, Barry. Software Engineering Economics, Prentice Hall, Upper Saddle River, NJ, 1981.
9. Иосиф Каршенбойм. Микроконтроллеры NEC для автомобильной электроники-2. // Компоненты и технологии. 2006. № 1.
10. Иосиф Каршенбойм. Микропроцессор своими руками-3. // Компоненты и технологии. 2006. № 3–4.
11. Юрасова Т. Поставщик дворов их глобальных величеств. tt p://w ww.expert.r u/rus_business/2006/03/interveiw_vorohovskiy/
12. RTI: The Economic Impact of Inadequate Software Testing, Planning Report 02-3, May 2002, ht tp:// w ww .nist.g ov/director/prog-ofc/report02-3.pdf.
13. Glass, Robert. Facts and Fallacies of Software Engineering, Addison-Wesley, New York, NY, 2002.
14. Davis, Alan. 201 Principles of Software Development, McGraw-Hill, New York, NY, 1995.