инженерия
программного
обеспечения

*Иванов Денис Юрьевич,*
*Новиков Фёдор Александрович*

# ВЛИЯНИЕ UML НА ПРОЦЕСС РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

**Аннотация**

В статье рассматривается вопрос о влиянии применения унифицированного языка моделирования UML на продуктивность процесса разработки программнго обеспечения. Рассмотрение опирается на предлагаемое авторами расширение известной модели инкрементного процесса разработки программного обеспечения новой концепцией циклов повышения продуктивности. Показано, что за счет применения UML возможно увеличение скорости движения информации в циклах повышения продуктивности и, тем самым, оказание положительного влияния на процесс разработки.

**Ключевые слова:** технология программирования, унифицированный язык моделирования, инкрементный процесс разработки.

В этой статье мы хотим поднять тему о влиянии систематического применения унифицированного языка моделирования (UML) на процесс разработки программного обеспечения.

Сам по себе UML не является моделью процесса разработки. UML – это графический язык моделирования общего назначения, предназначенный для спецификации, визуализации, проектирования и документирования всех артефактов, создаваемых при разработке систем, в том числе программного обеспечения [1]. Однако это определение не означает, что UML никак не связан с процессом разработки.

В этой статье мы хотим предложить свой ответ на вопрос, как влияет систематическое применение UML на процесс разработки. Для этого нам понадобится ввести в рассмотрение некоторые термины и понятия, после чего мы сформулируем наш основной тезис.

© Д.Ю. Иванов, Ф.А. Новиков, 2009

## ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ

Дисциплина, которая изучает процессы разработки программного обеспечения, по-английски называется Software Engineering. Для обозначения этой дисциплины мы будем использовать устоявшийся отечественный термин «*технология программирования*», хотя, может быть, этот термин немного устарел и не вполне точен, но нам так удобнее и привычнее. Технология программирования является инженерной дисциплиной, входящей в обязательный набор знаний и умений всякого инженера, причастного к созданию и эксплуатации программного обеспечения компьютеров. Технология программирования имеет четко выделенный объект изучения – процессы разработки и сопровождения программного обеспечения, но, в настоящее время, не имеет единого метода и общепринятого способа изложения. Технология программирования не является строгой математической дисциплиной, которую можно из-

ложить последовательно, начиная с основополагающих понятий и применяя дедуктивные доказательства. Напротив, технология программирования является собранием разнородных и часто несогласованных друг с другом моделей, методик и средств. Дадим несколько определений.

*Программирование* (computer programming) – это процесс создания программистом (человеком) программы (информационной структуры), предназначенной для последующего исполнения (компьютером).

Таким образом, в процессе программирования присутствуют явно субъект, объект и цель. В типичном (и привычном) случае субъектом является человек, который ведет процесс осознанно, объектом является текст на формальном языке, а целью является такое выполнение программы, которое в свою очередь имеет явно обозначенный результат.

*Технология программирования* (software engineering) – это совокупность методов и средств, позволяющих наладить производственный процесс создания программного обеспечения [2].

В этом определении особо следует подчеркнуть слово «производственный», которое отражает важнейшую особенность технологии программирования. Если производственный характер процесса разработки не требуется, то у технологии программирования фактически нет предмета.

Обсуждение вопросов технологии программирования в полном объеме выходит далеко за рамки этой статьи. Несмотря на то, что технология программирования начала развиваться одновременно с программированием, данная область знаний все еще находится скорее в стадии становления, нежели в стадии разработанной инженерной дисциплины, поэтому даже ее структура далеко не устоялась. У нас есть собственный взгляд на рациональный способ описания технологии программирования, которым мы воспользуемся для формулировки нашего тезиса о влиянии UML на практическое программирование.

Мы полагаем, что технологию программирования целесообразно рассматривать в трех аспектах.

- *Модель процесса*, то есть порядок проведения типового проекта по разработке программного обеспечения. Сюда относятся понятия жизненного цикла программного обеспечения, определение модели процесса – выделение в нем фаз, вех, потоков работ и других составляющих. Характерным для данного аспекта является рассмотрение на уровне программирующей организации в целом.
- *Модель команды*, то есть отношения между людьми в процессе разработки. Сюда относятся определение обязанностей работников – участников процесса, регламенты их взаимодействия, рабочие процедуры и т. п. Характерным для данного аспекта является рассмотрение на уровне группы (команды) или проекта.
- *Дисциплина программирования*, то есть методы создания конкретных артефактов, входящих в состав программного обеспечения. Сюда относятся описание и применение образцов проектирования, стандарты кодирования, методы тестирования и отладки и т. д. Характерным для данного аспекта является рассмотрение на уровне отдельного работника.

Все три составные части технологии программирования претерпели бурное и неравномерное развитие и продолжают постоянно меняться. Для понимания современных веяний в технологии программирования необходимо хотя бы кратко коснуться ее истории.

В истории технологии программирования происходило множество событий, выдвигалась и постоянно выдвигается масса новых идей. Чтобы как-то систематизировать эти факты, мы предлагаем следующую классификацию.

В первом приближении, все факты истории технологии программирования принадлежат одному из трех периодов.

***Дореволюционный период.*** С момента начала промышленной разработки программного обеспечения и до середины шестидесятых годов XX века вопросы собственно технологии программирования рассматривались, как правило, не отдельно, а

в связи и в совокупности с другими вопросами программирования. Разумеется, технологические проблемы существовали, и предлагались методы их решения, но это не было предметом публичных общественных дискуссий.

Компьютеры были дороги, их количество было невелико, и применялись они, в основном, для специальных целей (оборона, космос и т. п.). Следует подчеркнуть, что в это время программирование не являлось массовой профессий, и было, в основном уделом талантливых и высокообразованных одиночек, специалистов в той предметной области, где применялись компьютеры. Можно сказать, что хотя программирование и было высоко профессиональным, оно не было промышленным.

Из основных технологических идей, появившихся в этот период, следует отметить появление языков программирования и компиляторов, а также явное осознание важности модульного программирования как основы для накопления библиотек программ и их повторного использования.

***Революция в программировании.*** К середине шестидесятых годов XX века ситуация изменилась. Компьютеры стали дешевле, компактнее и производительнее. Сфера применения существенно расширилась, они стали в массовом порядке применяться в промышленности, науке, образовании. Наряду со сложными и ответственными программами (о самом существовании которых не везде можно было говорить вслух) появились, и в гораздо большем количестве, обычные программы для автоматизации производственной и иной повседневной деятельности.[1] Эти обычные программы изготавливались практически в каждой организации, имевшей компьютеры, независимо от основного профиля ее деятельности. Эксплуатация программного обеспечения перестала быть таинством, доступным только посвященным, программирование стало массовой профессией.

Заметим, что общество в целом не сразу осознало социальные последствия происходящего. В это время уже хорошо были известны положительные и отрицательные последствия других видов инженерно-технической деятельности. К проектированию таких изделий, как мосты и самолеты, любители уже не допускались (все понимали, что плохо спроектированный мост может обрушиться, а самолет упасть, и это недопустимо). Проектирование в традиционных инженерных областях велось в соответствии с многочисленными стандартами, правилами, регламентами и инструкциями, в которых число требований к качеству исчисляется сотнями и тысячами. Иное дело программирование: программисты в то время в большинстве своем и не слыхивали о каких-то стандартах качества своей работы. Да и действительно: подумаешь, программа расчета зарплаты «зависла» – это же не самолет упал!

Но низкая надежность – это только одна сторона проблемы. Хорошая технология не только улучшает качество, она еще и увеличивает производительность. А плохая технология – уменьшает. Отсутствие явно выписанной технологии – это самая плохая технология. В начале шестидесятых технология программирования, в современном понимании, отсутствовала как массовое явление. Реальная средняя производительность труда была низкой, что хорошо видно из отраслевых нормативов производительности программирования тех лет. Еще хуже дело обстояло с результативностью. Именно тогда были проведены первые методически обоснованные исследования и появились отчеты, из которых следовало, что менее половины проектов по разработке программ являются успешными. В средствах массовой информации появились мрачные прогнозы (полученные простой экстраполяцией наблюдаемых значений показателей), что к концу двадцатого века все трудоспособное население будет программировать, и программ будет не хватать. Кризис программирования был налицо.

[1] Сейчас такие программы называют бизнес-приложениями.

Очень быстро было предложено множество идей и подходов для выхода из кризиса.

Традиционно принято считать, что «первой ласточкой», положившей начало лавинообразному процессу сотворения технологии программирования, было письмо Э. Дейкстры в журнал Communications of the ACM в 1968 году, которому редактор, Н.Вирт, дал название «Go To Statements Considered Harmful[1]». Очевидно, что письмо Дейкстры подействовало как катализатор, как манифест – за несколько лет были опубликованы, обсуждены и практически внедрены следующие фундаментальные идеи технологии программирования:

– конструирование программ методом пошагового уточнения;

– проектирование сверху вниз и снизу вверх;

– структурное программирование;

– метод хирургической бригады;

– водопадная модель процесса разработки;

– жизненный цикл программного продукта.

По каждому направлению традиция называет основоположников (авторов наиболее замеченных из ранних публикаций), но фактически эти технологические идеи явились плодами совместных усилий, сотни людей внесли весомый вклад в это стремительное развитие.

***Послереволюционный период.*** Революция была успешной – в исторически кратчайшие сроки технология программирования оформилась как инженерная дисциплина, и некоторые ее положения повсеместно были внедрены в практику. Результаты проявились незамедлительно – средняя производительность труда в программировании увеличилась в несколько раз. Повысилась и надежность, за счет развития практики систематического тестирования. Кризис разрешился.

Развитее технологии программирования продолжалось, но уже не революционным, а эволюционным путем. Взятые за основу идеи 60-х годов развивались и совершенствовались.

Структурное программирование переросло в объектно-ориентированное, на смену водопадным моделям процесса пришли итерационные, метод хирургической бригады дал начало целому спектру моделей команды разработчиков. Но эти подходы не были революционно новыми, они улучшали и совершенствовали уже известное.

Однако время не стоит на месте. Аппаратное обеспечение стремительно прогрессирует. Сфера применения компьютеров расширяется все больше. Уже сейчас без них нигде нельзя обойтись, компьютеры применяются повсеместно. Пока еще слово «компьютер» ассоциируется у среднего жителя нашей планеты с образом персонального компьютера, с экраном и клавиатурой, а программирование ассоциируется с программированием на персональном компьютере и для персонального компьютера. Но это только пока. В самое ближайшее время компьютеры из устройств, которые используются очень часто и во многих местах, превратятся в устройства, которые используются везде и всегда. Речь идет о том, что сейчас называется встроенными системами. Персональные компьютеры выпускаются миллионами штук в год. Встроенные системы выпускаются миллиардами штук в год, а будут выпускаться многими миллиардами! И все эти устройства требуют для своей работы программного обеспечения. Не следует думать, что программировать эти устройства легко. Напротив. Уже сегодня возможности компьютера, скрытого в вашем мобильном телефоне, намного превосходят возможности того компьютера, которым пользовался Дейкстра, когда писал свое знаменитое письмо. Вполне вероятно, что консервативного усовершенствования старых идей, которого пока хватало для организации программирования персональных компьютеров, уже не хватит для организации программирования необозримого парка встроенных систем. Тогда технологию программирования ждут новые, действительно революционные изменения. Необходимо быть готовым к этому.

---

[1] «О вреде оператора Go To» *(перевод с англ.)*.

## ПОВЫШЕНИЕ ПРОДУКТИВНОСТИ ПРОГРАММИРОВАНИЯ

Очевидно, что задачей технологии программирования является исследование и улучшение процессов разработки программного обеспечения. При этом можно преследовать различные цели, то есть улучшать процесс в различных направлениях. Приведем несколько примеров задач, исследуемых и решаемых технологией программирования.

- *Повышение надежности программного обеспечения*. Программы часто содержат ошибки – это трюизм для практикующих программистов. Пользователи возмущаются, но терпят, пока это возможно. Если же это терпеть невозможно (например, в приложениях, критически важных для жизни людей), то применяются специальные методы технологии программирования, позволяющие повысить надежность программ. Как правило, такие методы достаточно трудоемки и требуют значительных ресурсов.
- *Снижение совокупной стоимости владения программным обеспечением*. Программы довольно дороги – причем дорогой зачастую является не только и не столько разработка, сколько сопровождение, модификация, обучение пользователей. Некоторые методы технологии программирования позволяют за счет небольших дополнительных расходов на этапе разработки добиться значительного снижения затрат на этапе эксплуатации.
- *Повышение продуктивности программирования*. Под продуктивностью здесь мы понимаем объем[1] разработанного программного обеспечения в расчете на одного работника за единицу времени. Для малых и средних программирующих организаций данный показатель фактически определяет конкурентоспособность.

Мы остановимся именно на последней задаче, как наиболее насущной для нас и полагаем, для большинства наших читателей. За счет чего можно повысить продуктивность программирования? В соответствии с нашими взглядами на природу технологии программирования формальный ответ очевиден:

– за счет улучшения процесса;

– за счет организации команды;

– за счет совершенствования дисциплины программирования.

Сформулируем несколько утверждений относительно возможности повышения продуктивности за счет применения различных методов. Сразу оговоримся, что эти утверждения основаны на личных наблюдениях авторов и не претендуют на статус всеобъемлющих законов природы.

Большинство программирующих организаций начинают борьбу за повышение продуктивности с формализации и документирования процесса. Мы считаем, что начальное введение формального процесса снижает продуктивность. Реальный рост продуктивности наблюдается только тогда, когда формализация процесса достигает достаточно высокого уровня зрелости, а именно, когда процесс является измеримым, а значит, управляемым. В любом случае, за счет совершенствования процесса продуктивность программирования удается увеличить на проценты, но не в разы и не на порядки.

Эффективная организация работы команды может дать существенно больший выигрыш в продуктивности и с меньшими затратами. Однако, аналогично начальной формализации процесса, начальное введение иерархии подчиненности и формализация должностных обязанностей, по нашему мнению, снижают продуктивность. Продуктивность существенно возрастает, если применяются тонкие и сложные методы организации команды, такие как динамическое переназначение работников и управление по компетентности. Динамическое переназначение подразумевает, что данный сотрудник участвует, как правило, в нескольких проектах одновременно и на разных фазах выполняет разные обязанности. Идея состоит в том, чтобы каждый сотрудник делал не «то, что полагается», а

[1] Проблема выбора измеряемых величин и единиц измерения для программ заслуживает отдельного обсуждения.

то, что он умеет делать лучше всех. Управление по компетенции организовать еще труднее: каждое решение должно приниматься не «начальником», а наиболее компетентным именно в данном вопросе сотрудником.

Но, по нашему мнению, главным резервом для повышения продуктивности является дисциплина программирования. Программирование вообще является редким примером области человеческой деятельности, где разброс индивидуальной продуктивности на порядок является скорее правилом, нежели исключением. Действительно, трудно представить себе, что один землекоп может стабильно и постоянно копать канавы в 10 раз быстрее другого. Но мы все знаем, что не такая уж редкость: программист, который стабильно работает в 10 раз лучше среднего, а бывают случаи, что и в 100 раз лучше!

Мы исходим из следующей качественной оценочной формулы, полученной чисто эмпирическим путем, в процессе наблюдений и размышлений над своим опытом и опытом ближайших коллег. Если для данной организации инвестиции в совершенствование процесса разработки в заданном размере $a$ дадут прирост продуктивности в $x$ %, то те же инвестиции в улучшение командной работы дадут прирост продуктивности в $3x$ %, а инвестиции в дисциплину программирования дадут прирост продуктивности в $10x$ %! Разумеется, мы не можем это доказать математически строго, но убеждены в справедливости этого правила.

Мы считаем, что имеются два фактора, решающим образом влияющих на продуктивность программирования:

– сокращение объема внеплановых изменений артефактов;

– увеличение объема повторно использованных артефактов.

Внеплановые изменения возникает при исправлении ошибок. Причем наиболее болезненны, как хорошо известно, ошибки, допущенные на ранних фазах, то есть ошибки в требованиях, ошибки проектирования. Речь идет не только и не столько об ошибках в коде программы – этот тип ошибок как раз сравнительно легко обнаружить и исправить. Туманное техническое задание, неудачное архитектурное решение, или плохой план пользовательской документации – примеры более серьезных ошибок. Изменения кода, вызванные изменениями требований, также, очевидно, уменьшают продуктивность. Однако это принципиально разные изменения. Грубо говоря, ошибки исправляются за счет разработчика, а изменения вносятся за счет заказчика. Поэтому исправление собственных ошибок всегда снижает продуктивность в денежном выражении, а переработка программного обеспечения по новым требованиям может и не сказываться на финансовой эффективности программирующей организации.

Повторное использование артефактов иногда наивно понимается как копирование текста из кода одной программы в код другой. Такая практика, разумеется, полезна, но на продуктивность практически не влияет. Все, что удается сэкономить – это время на ввод текста. Но программисты хорошо знают, что время на ввод текста программы едва ли превышает 1% от общего времени ее разработки. Повторное использование компонентов (модулей, классов) также сопряжено с трудностями: либо в момент создания компонента необходимо приложить заметные дополнительные усилия на подготовку к повторному использованию, либо компонент окажется неготовым и повторно использовать его не удается. Впрочем, по нашему мнению, наличие собственных заготовок для повторного использования (инструментов, библиотек, компонентов в репозитории) является главным признаком зрелости программирующей организации, поскольку в этом случае повторно используется голова, а не руки программиста.

Мы подошли к формулировке тезиса о влиянии UML на процесс разработки программного обеспечения. Рассмотрим рис. 1. На этом рисунке мы изобразили модель процесса разработки. С основным центральным циклом последовательного выполнения фаз процесса разработки сопряжены два внешних цикла движения определенных

артефактов. Правый верхний цикл, в который вовлечены заказчики, отражает выявление несоответствий требованиям и, тем самым, определяет объем внеплановых изменений. Левый нижний цикл, в который вовлечены разработчики, отражает преобразование разработанных артефактов в готовые к повторному применению элементы, тем самым, определяет объем повторного использования.

Теперь мы готовы сформулировать основной тезис данной статьи: средства для унификации представления информации в циклах повышения продуктивности позволяют сделать процесс разработки программного обеспечения более эффективным. Унификация обеспечивает ускорение прохождения циклов, повышение сохранности, облегчение восприятия и повышение надежности принимаемых решений. Надо ли говорить о том, что UML, как ни что другое, лучше всего подходит для решения таких задач [3].

И в завершении мы хотим дать несколько советов по внедрению UML в программирующих организациях.

## СОВЕТЫ ПО ВНЕДРЕНИЮ UML

***Организация должна поставить измеримую цель внедрения UML.*** Многие недооценивают важность постановки именно измеримой цели, и это грубая ошибка. Не стоит внедрять UML ради внедрения, только ради того, чтобы хвастаться потом этим перед коллегами. Это ничего хорошего не даст, последствия внедрения будут отрицательными, а не положительными. Мало того, что затраты на внедрение не окупятся ростом продуктивности, так еще и идея будет дискредитирована, и в будущем, при повторных, даже более подготовленных попытках, процесс внедрения пойдет более тяжело. Поставить измеримую цель не так просто. Для этого нужно иметь навыки и методики проведения измерений значений показателей процесса разработки (*метрик процесса*). В технологии программирования определено и используется великое множество метрик. Применяйте те метрики, которые лучше всего определены и точнее всего измеряются в конкретной организации.

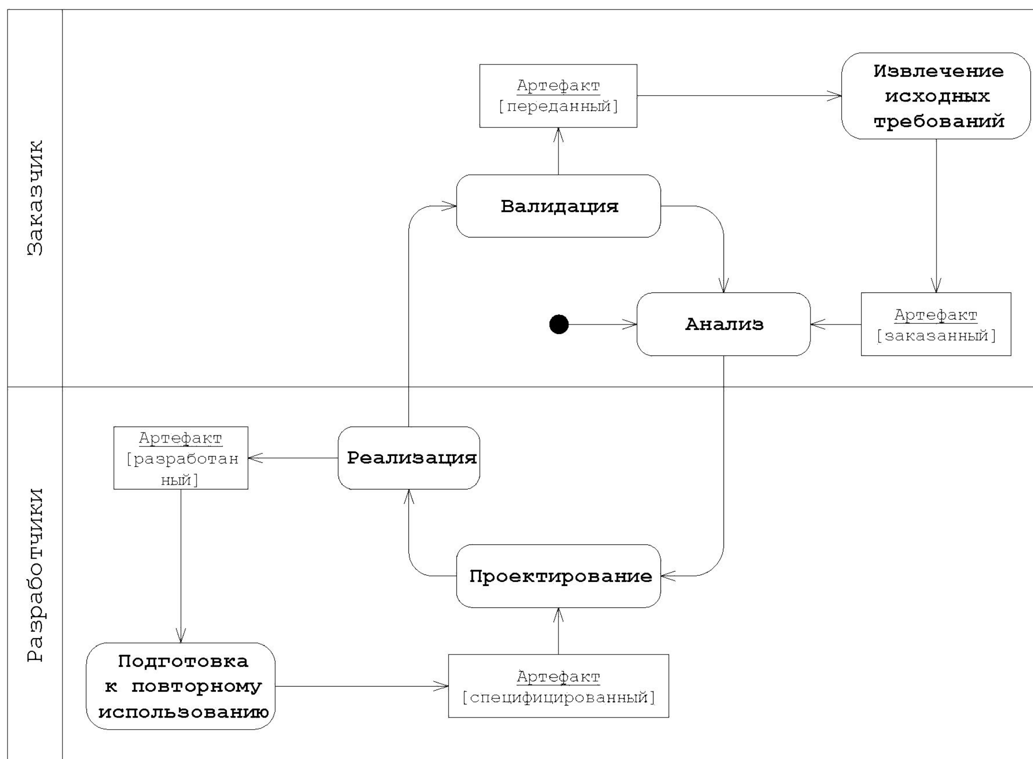


**Рис. 1.** Циклы повышения продуктивности

Приведем пример из личного опыта. Лично мы наиболее точно и надежно умеем измерять индивидуальные трудозатраты (в человеко-часах). Мы знаем, например, сколько раз была перерисована каждая диаграмма в нашей книге[1], и сколько это заняло времени. Имея в запасе индивидуальные показатели производительности при моделировании и проектировании, на их основе можно определить индивидуальные показатели при реализации и тестировании в расчете на один вариант использования. Проектируя систему уровня информационной системы отдела кадров или выше, полезно сделать прикидочный расчет трудоемкости. Нужно взять общее количество вариантов использования, умножить на эмпирический коэффициент и получится априорная оценка трудоемкости проекта. Когда проект будет завершен, нужно сравнить фактическую трудоемкость с априорной оценкой. Этот показатель называется точностью априорной оценки. Он очень важен для планирования и управления деятельностью программирующей организации. Так вот, можно поставить такую измеримую цель внедрения UML: повысить точность априорной оценки трудоемкости проектов до 90 %.[2]

***Знать и применять UML должны все участники процесса.*** Если в программирующей организации имеются один или два энтузиаста, которые пытаются описывать свои решения на UML, а большинство над ними снисходительно посмеивается, то такое применение ни на что не повлияет.[3] Если заказчик не может или не хочет верифицировать модель на ранних стадиях проектирования с целью выявления неправильно понятых требований, то толку не будет. Положительное влияние UML оказывается значительным, только если язык применяется массовым и систематическим образом. Особенно критичным является фактор вовлеченности высшего руководства организации в процесс внедрения UML. Если все высшее руководство поддерживает внедрение личным применением UML в подходящих случаях – отлично, цель, наверное, будет достигнута! А если среди руководства есть люди, которые отказываются визировать спецификации с диаграммами, потому что «не царское это дело», то, скорее всего, ничего не получится. Единственный надежный выход – провести квалифицированное корпоративное обучение методам применения UML [4].

***Должен использоваться корпоративный репозиторий решений на UML.*** Фактор повышения продуктивности зависит от повторного использования решений, специфицированных на UML. Но решения не используются повторно мгновенно – только через некоторое время. Значит, они должны где-то и как-то храниться. Такое хранилище называется *репозиторием* (repository). К великому сожалению, в данный момент мы никаких подходящих инструментальных средств для организации корпоративного репозитория на UML не знаем. Сами мы храним свои решения в обычной системе управления конфигураций, и держим в голове, что это именно решение на UML, и догадываемся по названию, для чего оно нужно. Для двух авторов это допустимо – количество наших хранимых решений измеряется двузначным числом. Но корпоративный репозиторий на организацию с сотней разработчиков должен будет содержать уже как минимум четырехзначное число хранимых элементов. Такой объем в уме не сохранишь и по названиям не догадаешься. Нужны специализированные и эффективные средства. Пока каж-

---

[1] Речь идет о нашей книге «Моделирование на UML», которую издательство «Наука и техника» планирует выпустить в январе 2010 года.

[2] На самом деле это очень агрессивная цель. Она означает, что если до начала проекта организация оценивает свои трудозатраты величиной $T$, то после окончания проекта фактические трудозатраты окажутся в интервале $[0.9T : 1.1T]$. Мало какие из программирующих организаций реально имеют такую точность планирования. Но внедрение UML позволяет этого достичь. Нам удавалось.

[3] Если не считать того, что многие люди крайне тяжело переживают ситуацию, когда к ним относятся как к «городским сумасшедшим».

дый изобретает как умеет. И вам приется этим заняться, если требуется внедрить UML в вашей организации.

Мы сами по нескольку раз пережили процесс внедрения UML, причем с обеих сторон: и как те, *в кого* внедряют UML, и как те, *кто* внедряет UML. Надеемся, что наши простые советы, основанные на выводах из сделанных ошибок, могут оказаться полезными.

**Литература**

1. *Буч Г., Якобсон А., Рамбо Д.* UML. 2-е издание Классика CS. СПб.: Питер, 2005.

2. *Терехов А.Н.* Технология программирования. Бином, 2007.

3. *Новиков Ф.А.* Визуальное конструирование программ // Информационно-управляющие системы, 2005. № 6. с. 9–22.

4. *Иванов Д.Ю., Новиков Ф.А.* Моделирование на UML: опыт преподавания в Интернет // Компьютерные инструменты в образовании, 2009. № 4. С. 53–61.

**Abstract**

The influence of Unified Modeling Language onto software development process is considered. Authors introduce new concept of productivity increasing cycles into well-known incremental software development process model and base argue upon this innovation. We show, that using UML one can gain the increasing of velocity of information transition within productivity cycles and thus yield increasing of productivity.

***Иванов Денис Юрьевич,***
***ИТ-консультант, Ай Ти Консалтинг,***
***denis.ivanov@it-konsulting.spb.ru***

***Новиков Фёдор Александрович,***
***кандидат физико-математических наук, заведующий лабораторией астрономического программирования ИПА РАН,***
***fedornovikov@rambler.ru***

Наши авторы, 2009.
Our authors, 2009.