anylogic

ПРИМЕНЕНИЕ ИМИТАЦИОННОГО МОДЕЛИРОВАНИЯ ПРИ УПРАВЛЕНИИ КОМПЛЕКСНЫМИ ПРОЕКТАМИ

Глава из книги

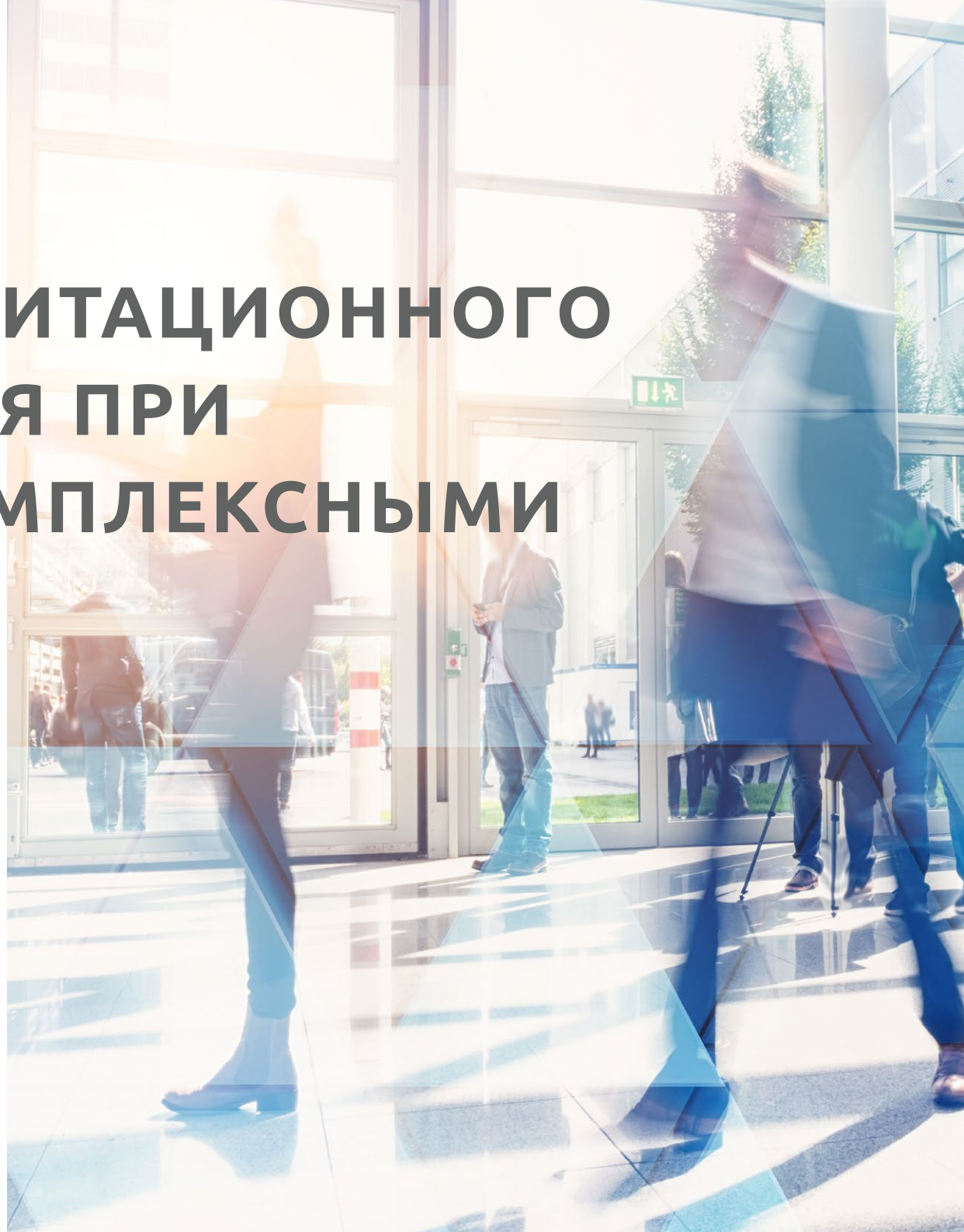
Modeling and Simulation in Complex Project Management

СЕРГЕЙ СУСЛОВ

Магистр со специализацией «Компьютерное моделирование», директор по международным продажам и маркетингу, Компания AnyLogic

ДМИТРИЙ КАТАЛЕВСКИЙ

Магистр в области управления Государственного университета штата Нью-Йорк (MPA), кандидат экономических наук, директор департамента индустриальных программ Сколковского института науки и технологий, преподаватель Института бизнеса и делового администрирования РАНХиГС



anylogic

ОГЛАВЛЕНИЕ

ВСТУПЛЕНИЕ

эта глава книги Modeling and Simulation in Complex Project Management посвящена современным методам имитационного моделирования, которые применяются при управлении комплексными проектами. Глава состоит из двух частей:

- В первой части приведен **теоретический обзор управления проектами с точки зрения системного подхода с применением причинно-следственных диаграмм** — методики, которая часто используется для получения концептуального представления о комплексных системах. В этой части отражены типичные причины провалов проектов. Для иллюстрации того, как может выйти из-под контроля комплексный проект, использован пример из реальной практики компании-разработчика программного обеспечения. Приведен краткий обзор литературных источников по исследованиям, посвященным применению имитационного моделирования при управлении проектами, а также даны практические советы менеджерам.
- Во второй части главы приведен **обзор наиболее широко применяемых методов моделирования (системно-динамический, агентный, дискретно-событийный) с использованием некоторых простых моделей в качестве примеров**. Эта часть может быть интересна тем, кто делает первые шаги в имитационном моделировании проектов. Приведенные здесь примеры выполнены с использованием ПО AnyLogic.

ИМИТАЦИОННОЕ МОДЕЛИРОВАНИЕ

ИМИТАЦИОННОЕ МОДЕЛИРОВАНИЕ

Модели играют важную роль в нашей жизни, хотя люди не всегда отдают себе отчет в том, как часто их используют. Простой подсчет того, сколько времени и денег вам понадобится, чтобы попасть из отеля в центр города во время путешествия, – это уже использование ментальной модели. Такая модель даже может содержать уравнения, например, когда вы умножаете цену билета на количество членов вашей семьи. Модели постоянно встречаются в нашей повседневной жизни. Они могут принимать различные виды, такие как ментальные модели, формулы, блок-схемы, физические модели зданий и так далее.

Моделирование в цифровой среде с применением средств программного обеспечения стало неотъемлемой частью современного предпринимательства и проектирования. Во многих областях для управления комплексными проектами применяются специализированные средства программного обеспечения. Хорошим примером является моделирование в строительной отрасли – информационное моделирование зданий. На сегодняшний день перед началом строительства создаётся цифровая модель здания с помощью соответствующего ПО, например, Autodesk Revit. Такие модели включают все архитектурные и инженерные характеристики будущего здания (на рис. 1 приведен пример)

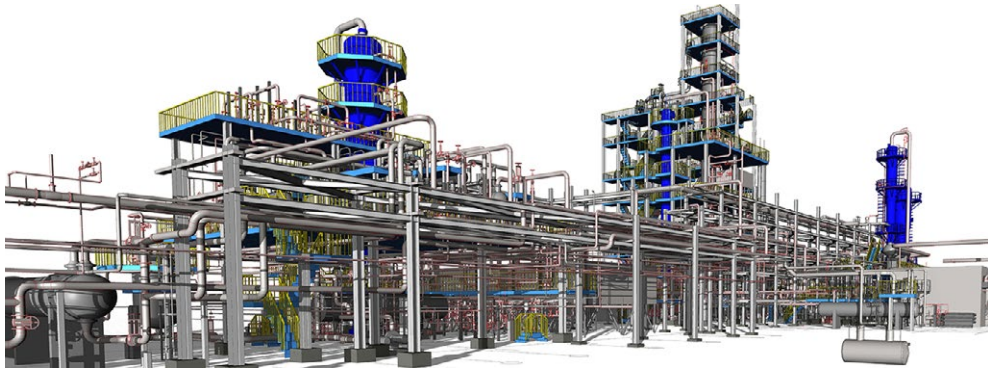


рис. 1 ИНФОРМАЦИОННАЯ МОДЕЛЬ ЗАВОДА ПО ПЕРЕРАБОТКЕ НЕФТИ, ВЫПОЛНЕННАЯ В AUTODESK REVIT [1]

Подобные модели впоследствии облегчают строительство, поскольку все задействованные в сооружении объекта специалисты (архитекторы, инженеры-проектировщики, строители), а также будущие работники построенного здания (технологи, занимающиеся автоматизацией завода, или работники здравоохранения, организующие работу больницы) с помощью модели могут разобраться в технической стороне проекта.

Почему люди создают модели? Прежде всего, потому, что проводить эксперименты с моделью легче, чем с реальным объектом. Реальные эксперименты часто бывают слишком дорогими или даже физически

1 - Иллюстрация модели с сайта

невыполнимыми. Например, горизонт планирования строительства нового морского порта может составлять десятки лет. Невозможно экспериментально проверить, как лучше строить такой крупный объект, если существует несколько вариантов строительства. Однако это можно сделать с помощью модели.

Мир виртуальных моделей свободен от каких-либо рисков. Менеджеры, исследователи и инженеры могут создавать и тестировать модели различной конфигурации и с помощью виртуальных экспериментов в безрисковой среде проверять сотни своих гипотез. Имитационное моделирование – это эффективный инструмент улучшения взаимодействия участников на ранних стадиях комплексных проектов, поскольку оно дает

рис. 2 БЕЗРИСКОВЫЙ ВИРТУАЛЬНЫЙ «МИР» МОДЕЛЕЙ

почву для объективных обсуждений. Не приходится судить о ценности мнения специалиста на основании его прежнего опыта и заслуг. Менеджеры и инженеры могут приводить результаты моделирования, чтобы подкрепить свои предложения.

Имитационное моделирование отличается от остальных видов и технологий моделирования тем, что позволяет пользователям создавать динамические модели. Такая модель со временем развивается, обновляя свой статус постоянно либо с заданной периодичностью. Создание имитационной модели предусматривает определение набора правил, в соответствии с которыми она будет изменяться во времени.

Любая имитация объекта – это модель, но не каждая модель является имитационной. К примеру, наиболее популярным инструментом для создания моделей являются электронные таблицы Microsoft Excel. Но можно ли считать подсчет бюджета проекта в Excel имитационной моделью? Конечно же, нет, так как в ней отсутствует временная компонента, отражены лишь статичные данные за определенные моменты времени.

Имитационное моделирование – это мощный инструмент, который широко используется для решения самых разных бизнес-задач. Метод популярен благодаря своей простоте, понятности и удобству. С ним не нужно строить предположения наугад, можно просто запустить имитационную модель и посмотреть, что произойдет. Еще одно значительное преимущество имитационных моделей по сравнению со статичными (например, разработанными в Excel) – поддерживаемая ими анимация. Когда люди наблюдают на видео, как система, например, порт, будет работать при заданных параметрах и конфигурации оборудования, им проще визуально убедиться в правильности работы модели, и они больше доверяют ее результатам.

Еще одним значительным преимуществом имитационных моделей является учёт стохастики. Любую характеристику имитационной модели можно задать как вероятностное распределение с заданными параметрами, к примеру, длительность стадии проекта. При каждом запуске модели этому параметру будет присваиваться случайное значение, сгенерированное по заданному вероятностному распределению. Если модель запускается множество раз, то результат моделирования (например, общее время реализации проекта) можно рассматривать как стохастическую величину, построить для неё гистограмму или диаграмму типа «ящик с усами», вычислить среднее значение и отклонения от него. Это позволяет учитывать отклонения, которые отражают риск перерасхода времени или бюджета.

Изображенная ниже диаграмма типа «ящик с усами» показывает результаты имитационного моделирования четырех сценариев, отражающих четыре разные стратегии. Эта диаграмма позволяет сравнить стратегии, учитывая не только средние показатели производительности (чистый дисконтированный доход, длительность реализации проекта), но и отклонения. «Ящик» отображает область значений пятидесяти процентов результатов, а «усы» – всю область значений результатов (Компания AnyLogic, 2016).

Время и стохастика – это основные два элемента, которые делают имитационные модели наиболее эффективными для использования в управлении комплексными проектами. Поэтому далее будут рассматриваться исключительно имитационные модели.

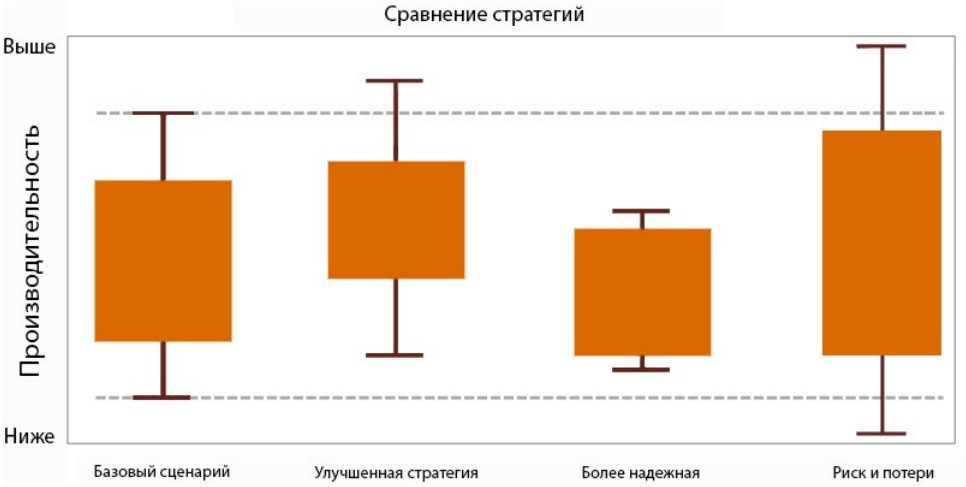


рис. 3 ДИАГРАММА ТИПА «ЯЩИК С УСАМИ», ОТРАЖАЮЩАЯ РЕЗУЛЬТАТЫ ИМИТАЦИОННОГО МОДЕЛИРОВАНИЯ ЧЕТЫРЕХ РАЗНЫХ СТРАТЕГИЙ[1]

1 - Иллюстрация из презентации Лайла Уоллиса, директора PricewaterhouseCoopers Advisory Services LLC. Подробнее об опыте PwC US

ПРИМЕНЕНИЕ ИМИТАЦИОННЫХ МОДЕЛЕЙ ПРИ УПРАВЛЕНИИ ПРОЕКТАМИ

02

ПРИМЕНЕНИЕ ИМИТАЦИОННЫХ МОДЕЛЕЙ ПРИ УПРАВЛЕНИИ ПРОЕКТАМИ

Изучение возможности использования имитационного моделирования для работы с комплексными проектами имеет многолетнюю и богатую историю. Так, первые исследования в области имитационного моделирования проводились в 1960-х годах, однако лишь в 1980-х они начали применяться в работе с конкретными проектами. С 1990-х начали выходить многочисленные научные публикации, описывающие использование имитационного подхода при управлении проектами. Многие исследования по управлению проектами и имитационным моделям были посвящены поиску основных причин срыва проектов и значительных перерасходов средств, которые нередко имели место.

В первых систематических исследованиях Робертс применил имитационные модели (в частности, системно-динамические) для управления проектами, введя понятие потоков работ по проекту (единиц деятельности) и концепцию разрыва между восприятием и реальностью (разницы между предполагаемым и реальным прогрессом). В своих исследованиях Робертс указывал на то, что объем работ и усилий, необходимых для успешного завершения проекта, часто недооценивается и у ответственных лиц складывается искаженный взгляд на проект, что, в свою очередь, неминуемо ведет к нерациональному использованию ресурсов (Roberts, 1964, 1974). С помощью первых созданных моделей Робертс изучал *разницу между предполагаемым и реальным прогрессом* в выполнении научно-исследовательских проектов, динамику их развития, а также эффективность мультипроектного управления. Это были ранние попытки исследовать влияние управленческих решений на осуществление проекта, основываясь на предположении, что статус проекта может отличаться от реального состояния работ.

Позднее Ричардсон и Пью разработали концепцию выявленных и невыявленных ошибок, предполагаемого прогресса и реальной продуктивности (в данный момент эти труды считаются классикой в области системно-динамических моделей для управления проектами). В модели, представленной Ричардсоном и Пью, особое внимание уделялось изучению нескольких ключевых понятий, свойственных любому комплексному проекту: цикл мониторинга и контроля, исправление ошибок и найм персонала (Richardson, Pugh, 1981).

Опираясь на достигнутые успехи в области имитационного моделирования проектов, Пью и Робертс создали «Систему моделирования для управления проектами» (набор сложных системно-динамических моделей, предназначенных для использования в качестве инструмента поддержки принятия управленческих решений). Инструмент успешно применялся

в ряде проектов по управленческому консалтингу в сфере крупного строительства и даже урегулирования споров (случаи возникновения задержек и сбоев в осуществлении проекта). Одним из первых успехов было урегулирование спора между кораблестроительной компанией Ingalls Shipbuilding и Военно-морскими силами США в 1970-х. Ingalls Shipbuilding выиграли контракт на строительство нескольких военных кораблей в 1960-1970-х годах. Контрактом была установлена твердая договорная цена, однако перерасход по смете составил 500 миллионов долларов. Заказчик согласился покрыть лишь 150 миллионов прямых издержек, а перерасход оставшихся средств считал следствием неэффективного управления проектом со стороны руководства Ingalls. Компания Ingalls, в свою очередь, подала в суд на ВМС США, утверждая, что причиной перерасхода стало постоянное внесение изменений в конструкцию. Фирма Пью-Робертса Pough-Roberts Associates разработала комплексную модель, которая детально воссоздавала кораблестроительный проект, чтобы оценить финансовые последствия инициированных заказчиком задержек в работе и изменений конструкции. В результате спор был урегулирован без судебного разбирательства и ВМС США выплатили Ingalls компенсацию в 447 миллионов долларов. Изучив этот случай, Купер определил, что примерно 200-300 тысяч из этой выплаты удалось получить благодаря анализу, проведенному с помощью модели (Cooper, 1980).

За прошедшие десятилетия еще многие исследователи также внесли свой вклад в развитие имитационного моделирования, используемого при управлении проектами:

- Абдель-Хамид (Abdel-Hamid, 1993) (интерактивная модель управления проектом по созданию программного обеспечения): интеграция управленческих функций планирования и подбора персонала для разработки программного обеспечения;

- Купер (Cooper, 1980, 1993a, 1993b, 1993c; Cooper et al., 2002; Cooper and Reichelt, 2004): управление проектами как комплексная система, нелинейная обратная связь, подсчет резонансного эффекта на перерасход средств;

- Вильямс и соавторы (Williams et al.,1995): влияние задержек и внесения проектных изменений на стоимость проекта (кумулятивный эффект);

- Форд и Стерман (Ford and Sterman, 1998): модель многоэтапного проекта (цепочечная структура с распределением по срокам), детально отображающая стадии работы по четырем видам деятельности в рамках проекта, чтобы дать целостное представление о ходе работ;

- Вильямс (Williams,1999): исследование факторов, осложняющих управление проектами;

- Грэхэм (Grapham, 2000): выводы, полученные при моделировании комплексных программ системно-динамическим методом;

- Линейс и соавторы (Lyneis et al., 2001): стратегическое управление комплексными проектами;

- Эден и соавторы (Eden et al., 2005): анализ исков, связанных с провалами комплексных проектов (сравнение системно-динамического моделирования и методов «мерной линии»);

- Форд и Тейлор (Ford and Taylor, 2006): стратегии кризисного управления проектами;

- К примеру, Ли (Lee et al., 2007) исследовал взаимосвязь задержек в распределении ресурсов и разных степеней контроля, которые вводили менеджеры проектов, и пришел к некоторым неожиданным выводам (например, что задержки неизбежны, но наиболее благоприятно, если они будут краткосрочными).

И этот список далеко не полный.

Опираясь на многочисленные исследования примеров неудачного управления проектами, практики имитационного моделирования разработали обобщенную *теорию механизма возникновения сбоев при осуществлении проектов.* Далее эта теория будет изложена на примере пошагового описания динамики возникновения проблем в проекте компании по разработке ПО.

2.1 ПРИМЕР ИЗ ПРАКТИКИ: МЕХАНИЗМ ВОЗНИКНОВЕНИЯ СБОЕВ В ОСУЩЕСТВЛЕНИИ ПРОЕКТА

Важно понимать разницу между традиционными методами управления проектами и имитационным моделированием (системно-динамическим, агентным методом и т. д.). При традиционном подходе основанные на методе критического пути инструменты управления проектом представляют собой проект в виде связанной последовательности отдельных технических задач и событий. Таким образом, при этом подходе проект является суммой отдельных составляющих работы. Такие инструменты и системы управления могут ввести в большое заблуждение, потому что не показывают, что в реальности *осуществление проекта – это не линейное последовательное выполнение ряда задач, а циклический процесс,* где каждая задача проходит несколько итераций. Более того, эти инструменты способствуют восприятию проектов в виде неуправляемых ракет, стремящихся к результату, на который мало влияет вмешательство человека (Cooper, 1994).

Среди практиков имитационного моделирования, в частности метода системной динамики, широко распространено мнение, что ключевой причиной провалов проектов является слишком сложная система обратной связи, затрудняющая понимание проектов ответственными лицами и принятие решений. Принятые решения и меры обычно приводят в действие запутанную систему множества причинно-следственных связей. Это, как правило, влечет за собой задержки, мешающие осуществлению проекта. В данной части публикации описывается типичный механизм срыва проектов, предложен ряд способов предотвращения подобного развития событий и советы по смягчению негативных последствий.

Представим, что некая компания ABC хочет осуществить комплексный проект по разработке программного обеспечения. Допустим, что для того, чтобы успешно выпустить ПО, сотрудникам компании ABC необходимо написать примерно 100 тысяч строк исходного кода.[1] Используя устоявшийся язык системно-динамического моделирования – диаграмму потоков и накопителей, мы можем представить структуру этого проекта следующим образом (рис. 4):

рис. 4 ВИЗУАЛЬНОЕ ОТОБРАЖЕНИЕ НАКОПИТЕЛЕЙИ ПОТОКА В МОДЕЛИ

Накопитель[2] **Запланированная работа** отражает изначальный объем работы, который, по мнению руководства компании ABC, необходимо выполнить, чтобы выпустить программный продукт (например, 100 тысяч строк кода). Накопитель **Выполненная работа** отражает ту часть работы, которая закончена и не нуждается в дальнейшей доработке. В обоих накопителях в качестве единицы измерения используется число строк программного кода. По мере разработки программного обеспечения **Запланированной работы** становится меньше, а **Выполненной** больше. Поток[3] **Выполняемая работа** отражает ход выполнения работ в конкретный момент времени. Значение потока может исчисляться в строках кода, добавленных за определенный период времени, например, за день.

Обычно на скорость **Выполняемой работы** влияют непосредственно **Сотрудники** и **Продуктивность** (рис 5). Можно было бы предположить, что чем больше технического персонала (разработчиков программного обеспечения) будет привлечено к работе над ПО и чем продуктивнее они добавляют строки к коду в заданный период времени, тем быстрее будет расти значение накопителя **Выполненная работа**. Однако разумно предположить, что не вся **Выполняемая работа** полезна и отвечает нашим требованиям к качеству. В систему необходимо включить новую переменную **Качество**, которое оценивает единицы **Выполняемой работы** (присваивает значение 1 или 0), прежде чем они будут добавлены в накопитель **Выполненной работы** и, соответственно, не будут нуждаться в дальнейшей доработке. Для простоты примем, что **Качество** связано с количеством ошибок, допущенных в программном коде. На самом же деле

1 - Продуктивность выполнения проекта разработки программного обеспечения может измеряться разными способами: строками исходного кода, написанными за определенный промежуток времени; количеством ошибок на тысячу строк кода; рабочими днями (потраченным программистом временем на написание кода без учета планирования и прочей сопутствующей основной работе деятельности); объемом кода, который программист может написать за год и т. д. Иногда в качестве меры продуктивности используют длительность цикла (от принятия в производство до выполнения) и срок выполнения (от получения заказа от клиента до передачи ему продукта). Здесь для удобства продуктивность измеряется в строках кода, написанных за определенное время (хотя многие специалисты в этой области считают этот способ измерения продуктивности разработки ПО неэффективным).

2 - **Накопитель** — элемент системы, который со временем наполняется или истощается. Накопители отображают состояние системы и могут быть изменены лишь Потоками.

3 - **Поток** — течение чего-либо из Накопителя или в Накопитель. Потоки отражают процессы – изменения в накопителях в любой заданный момент времени.

на **Качество** и **Продуктивность** может прямо либо опосредованно влиять множество факторов как внутренних, так и внешних.

Как отмечалось прежде, большим скачком в понимании динамики управления проектами было добавление **Цикла исправления ошибок**

рис. 5 ВЛИЯЮЩИЕ ПЕРЕМЕННЫЕ

(рис 6). Ошибки выявляются в процессе тестирования. По мере выявления ошибок (в зависимости от скорости их обнаружения), соответствующая часть кода отправляется на доработку и увеличивает объем работы, которую необходимо выполнить.

Обычно при создании системно-динамических моделей используется накопитель **Выявленные ошибки**, обозначающий объем работы, которую нужно переделать. Также существует понятие **Невыявленные ошибки**, обозначающее работу, содержащую существующие, но еще не выявленные ошибки. Значение накопителя **Выявленные ошибки** увеличивается в ходе входящего потока **Выявление ошибок** (в данном случае единица измерения – количество строк, классифицированных как нуждающиеся в исправлении, за единицу времени) и уменьшается за счет исходящего потока **Исправление ошибок** (количество исправленных строк кода за единицу времени). Выявление ошибок может вызывать определенные сложности, ведь они далеко не всегда быстро обнаруживаются. Исследование NASA, посвященное проблемам разработки средств программного обеспечения полета (NASA Office of Chief Engineer, 2009: 47) показало, что в процессе разработки программного обеспечения много ошибок допускается на стадиях создания архитектуры программы и написания кода (рис. 7).

Однако даже накопитель **Выявленные ошибки** может содержать ошибки, которые потребуют повторного прохождения цикла исправления ошибок.

рис. 6 ТИПИЧНЫЙ ЦИКЛ ИСПРАВЛЕНИЯ ОШИБОК

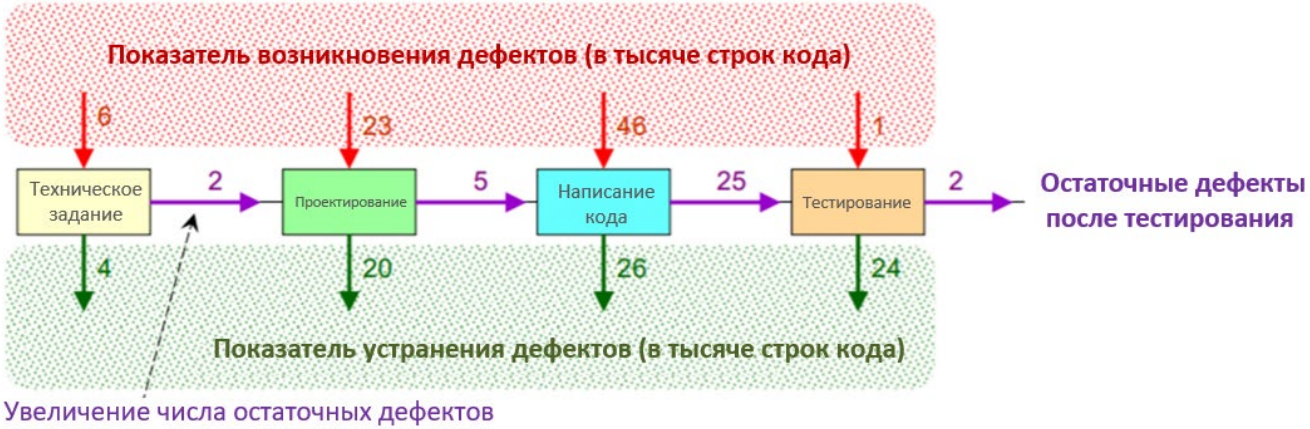


рис. 7 УВЕЛИЧЕНИЕ ЧИСЛА ДЕФЕКТОВ НА РАЗЛИЧНЫХ СТАДИЯХ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ (ПОСЛЕ ТЕСТИРОВАНИЯ ОСТАЕТСЯ 2 ДЕФЕКТА НА 1000 СТРОК КОДА) (EICK ET AL., 1992)

Как заявляет Холзман:

> Программное обеспечение системы управления, необходимой, допустим, для осуществления космического полета, насчитывает немногим меньше миллиона строк кода. Если мы обратимся к отраслевой статистике, то увидим, что качественный, дорогостоящий процесс разработки может уменьшить число ошибок в таком коде до примерно одного остаточного дефекта на 10 тысяч строк (остаточный дефект – ошибка, оставшаяся после того, как код был полностью протестирован и выпущен. Более многочисленные дефекты, скрытые в коде, часто называют латентными). Поэтому в работе системы с миллионом строк кода предположительно возникнет как минимум 100 дефектов
>
> *Holzmann, 2007*

Как отмечают Родригес и Боуэрс (Rodrigues and Bowers, 1996), в цикле исправления ошибок можно выделить четыре ключевых фактора, которые могут и должны контролироваться менеджерами проекта:

- необходимый объем ресурсов,
- продуктивность,
- качество выполнения работы,
- время выявления ошибок.

Обычно лица, управляющие проектом, уделяют наибольшее внимание ресурсам и продуктивности. В случае компании ABC – сотрудникам и их навыкам программирования, которых рассматривали как залог успеха или срыва проекта. Однако из опыта моделирования системно-динамических систем известно, что в равной степени важно отслеживать качество работы и **показатель выявления ошибок**.

Линейс и Форд называли цикл исправления ошибок характеристикой, оказывающей наибольшее влияние на успешность проекта:

> По нашему мнению, цикл исправления ошибок является наиболее важной характеристикой системно-динамических моделей

проектов. Повторяющийся характер цикла исправления ошибок, при котором исправления влекут за собой новые ошибки, снова требующие исправления и т. д., создает критические режимы работы, которые часто затягиваются почти на весь срок выполнения проекта и становятся причиной многих проблем.

Lyneis and Ford, 2007

Цикл исправления ошибок легко может занять половину времени осуществления проекта (Cooper, 1994).

Для дальнейшего анализа работы компании ABC над проектом по разработке программного обеспечения необходимо добавить причинно-следственные цепочки и петли обратной связи, отражающие динамику жизненного цикла проекта.

Руководство компании ABC контролирует реализацию проекта с помощью периодической оценки прогресса в работе. Прежде всего, они попытаются оценить необходимость привлечения дополнительного персонала, чтобы закончить оставшуюся работу (включая исправление выявленных ошибок) в срок. Так, **Предполагаемый прогресс** выражается в расчете **Ожидаемого срока завершения работ**. Если **Ожидаемый срок завершения** значительно отличается от **Назначенного срока завершения**, то руководству потребуется принять меры.

Наиболее распространенный общепринятый подход – *привлечь больше сотрудников*, чтобы увеличить объем **Выполняемой работы** (этот цикл отражен на рис. 8: *Невыявленные ошибки → Предполагаемый прогресс → Ожидаемый срок завершения → Назначенный срок завершения → Оставшееся время → Необходимость в дополнительном персонале → Привлечение персонала → Найм → Работа привлеченного персонала над проектом).*

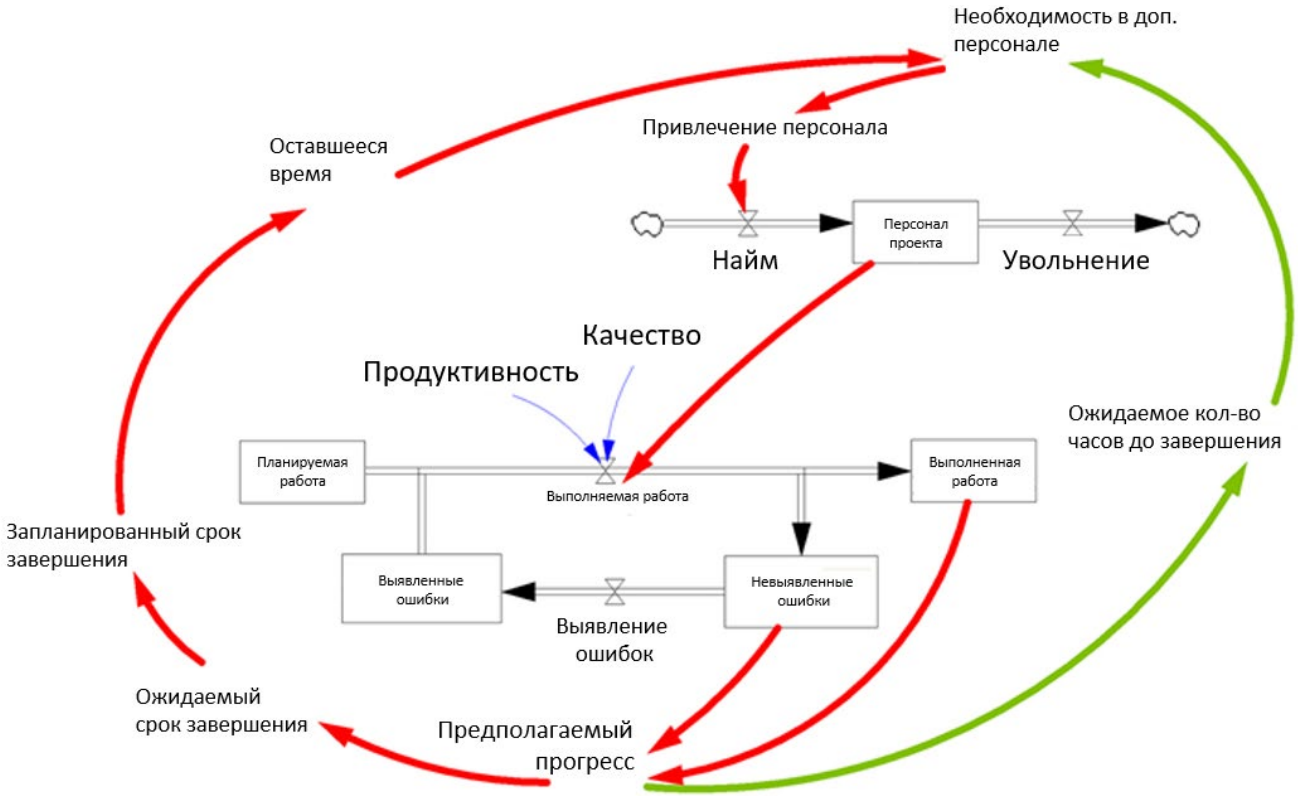


рис. 8 ЦИКЛ «НАМ НУЖНО БОЛЬШЕ ЛЮДЕЙ!»

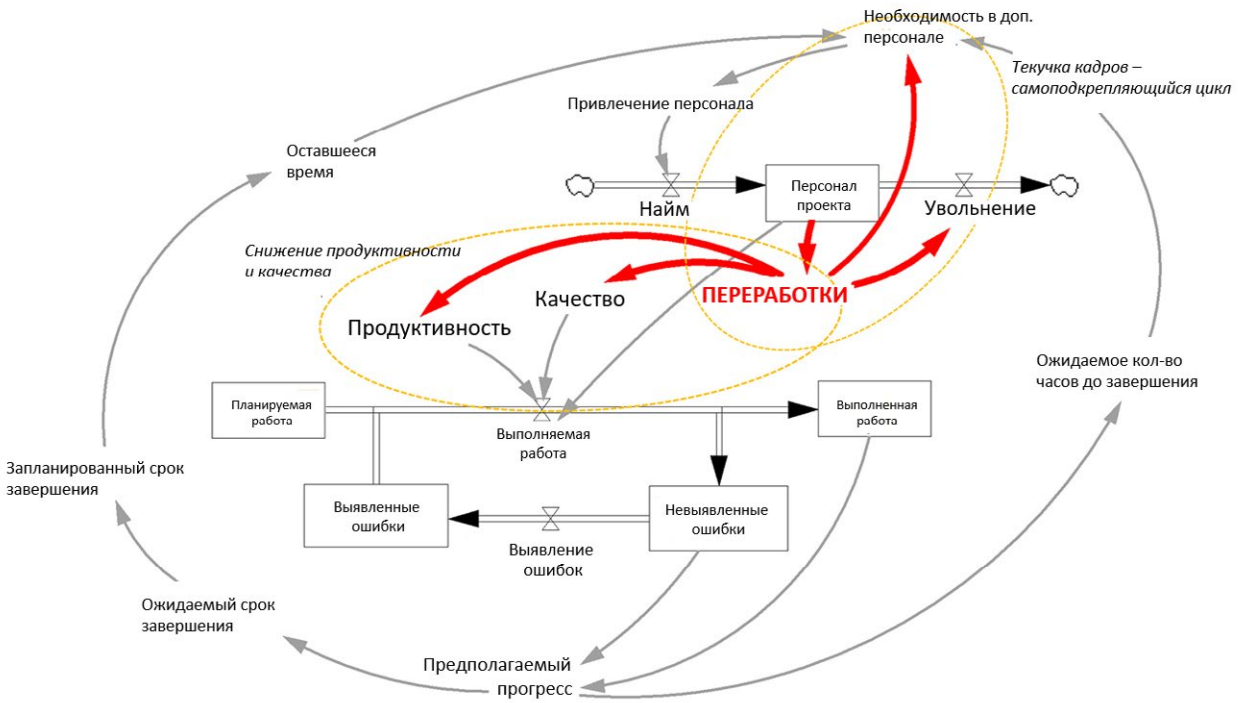


рис. 9а ЦИКЛ ПЕРЕРАБОТОК

Если этот вариант по каким-либо причинам невозможен, например, ограничены ресурсы или недоступны кандидаты с подходящей квалификацией и навыками, то руководство введет **Продленный рабочий день** для группы разработки программного обеспечения (переработки). Это самый распространенный способ избежать дополнительных затрат и обязательств, связанных с наймом новых сотрудников.

Переработки быстро становятся для компании «обычной» новой практикой. Но разрыв между **Выполненной работой** и **Запланированной работой** не исчезает. Постоянные переработки через некоторое время снижают продуктивность и повышают количество совершаемых ошибок (эффект выгорания). Увеличение числа ошибок еще сильнее растягивает цикл исправления ошибок, что, в свою очередь, увеличивает объем работы и откладывает ее срок завершения. Весь цикл повторяется снова. Петля обратной связи от **Переработок** станет *самовоспроизводящейся*[1] (рис. 9а). Обычно руководство серьезно недооценивает цикл **Переработок** и его негативное влияние на работоспособность и моральное состояние персонала (Chan, 2011; Neves et al., 2016).

Через несколько недель несколько разработчиков программного обеспечения (фактически ключевые сотрудники) решают покинуть компанию ABC из-за постоянной **Работы в напряженном режиме** и подрыва **Морального духа** (рис. 9б). Часто наиболее ценные специалисты уходят первыми, поскольку с их навыками и квалификацией проще найти работу с лучшими условиями. Помимо того, что проект сталкивается с утечкой мозгов,

1 - **Самовоспроизводящаяся петля обратной связи (reinforcing loop)** — петля обратной связи, в которой суммарный эффект причинных связей со временем усиливает (подкрепляет) движение переменных величин в едином направлении за счет позитивной обратной связи (изменения в одном направлении ведут к еще большим изменения в том же направлении).

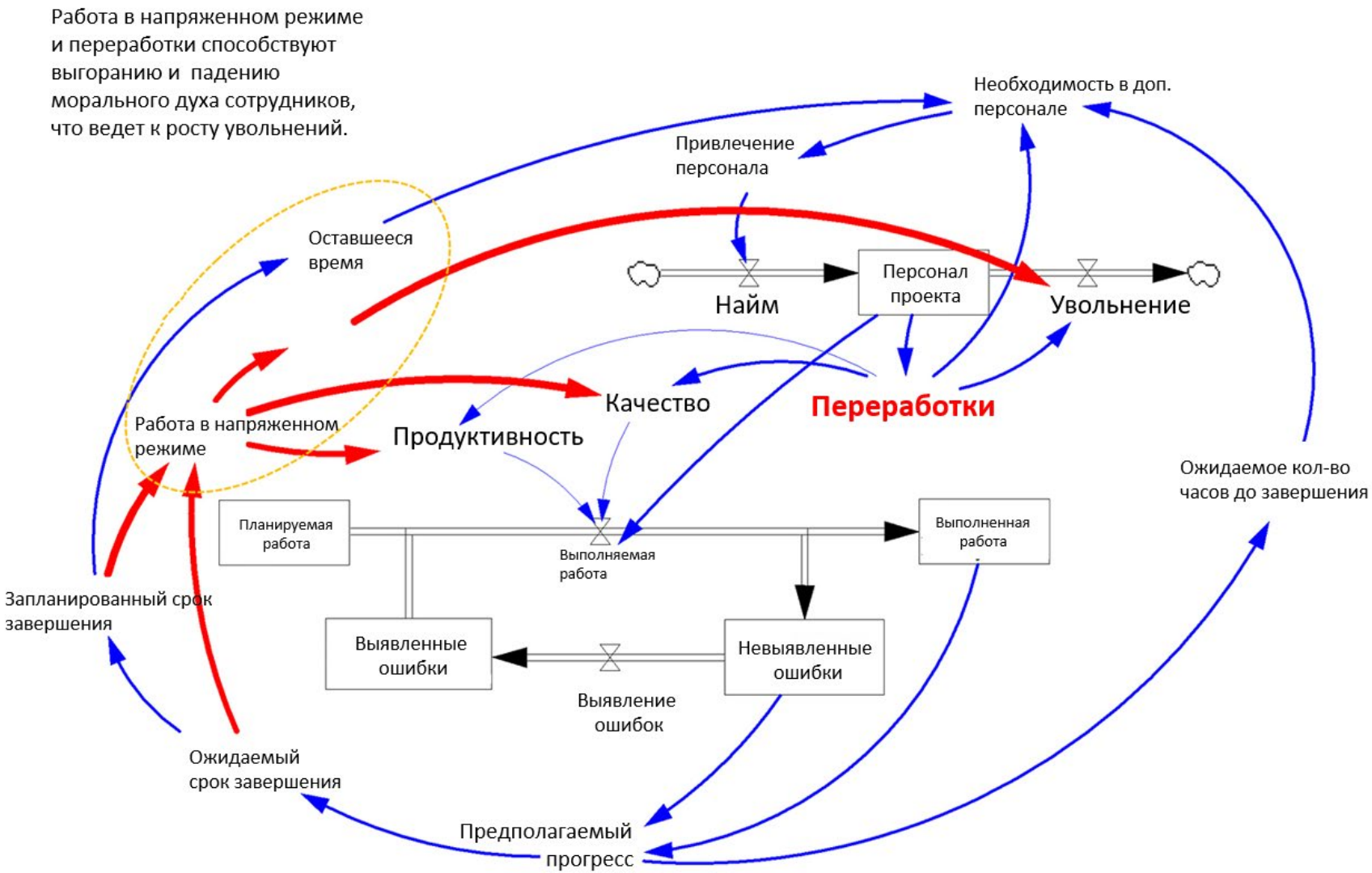


рис. 9б РАБОТА В НАПРЯЖЕННОМ РЕЖИМЕ И ПОДРЫВ МОРАЛЬНОГО ДУХА СПОСОБСТВУЮТ ТЕКУЧЕСТИ КАДРОВ

Запланированная работа уволившихся сотрудников перераспределяется между оставшимися сотрудниками, что дополнительно ухудшает их производственные показатели и усиливает самовоспроизводящуюся петлю обратной связи **Переработок**.

Теперь руководство намерено нанять новый персонал или привлечь к проекту своих сотрудников, ранее в нем не участвовавших. Однако на поиск и прием на работу подходящих специалистов требуется время (это отложенный процесс). Поэтому перегруженный оставшийся персонал продолжит падать духом, и **Качество работы** — снижаться.

Однако привлечение нового персонала имеет еще один парадоксальный эффект[1], который обычно недооценивают или игнорируют. Когда к работе в группе по разработке программного обеспечения привлекают новых сотрудников, проект пополняется менее опытными специалистами, чем те, кто уже был задействован в нем. Это в наибольшей степени касается случаев, когда нужны узкопрофильные специалисты или те, на которых спрос на рынке труда очень высок. Чем сильнее ограничен рынок труда, тем ниже уровень компетенций привлекаемых к проекту сотрудников (Рис. 10а). Поэтому новичкам требуется время, чтобы ознакомиться с проектом и пройти обучение, которое уже прошел остальной персонал.

Привлечение нового персонала вызывает возникновение ряда взаимосвязанных петель обратной связи. Новички могут преднамеренно или непреднамеренно способствовать дестабилизации коллектива за счет того, что иногда они менее преданы компании; могут не прижиться на новой работе; менее квалифицированы, чем предполагалось (переоценены); обладают нереалистичными представлениями о проекте и т. д. Это еще один порочный (самовоспроизводящийся) *цикл найма – увеличения числа увольнений – повторного найма.*

Новым сотрудникам нужно четко ставить задачи, их работа требует больше курирования и контроля. Поэтому квалифицированный персонал теперь уделяет все больше внимания не разработке программного обеспечения, а работе с новичками: обучению, помощи и контролю их работы (рис. 10б). Руководство компании ABC понимает, что в процессе разработки программного обеспечения теперь возникает еще больше ошибок, а цикл **Исправления ошибок** растягивается. Неожиданным результатом значительного расширения штата становится *снижение уровня профессионализма команды*, что негативно влияет на продуктивность и качество (рис. 10а-б). Снижается объем **Выполняемой работы**, **Качество** и **Продуктивность**, а число ошибок и объем работы по **Выявлению ошибок** и **Исправлению ошибок** возрастает. Разрыв между **Выполненной работой** и **Предполагаемым прогрессом** увеличивается, но на практике это обычно приводит к решению *нанять еще больше сотрудников!*

Команда менеджеров не может справиться с увеличением стоимости проекта, разрастающаяся структурная организация проекта не может нормально функционировать. Это влечет за собой проведение многочисленных собраний и обсуждений, но причины сложившегося положения остаются неявными. Еще один парадокс заключается в том, что, несмотря на увеличение состава проектной группы, доля полезного участия каждого из ее сотрудников может уменьшиться, поскольку теперь коллектив разделен на подгруппы и гораздо больше времени уходит на

1 - **Парадоксальное поведение** комплексных систем — термин, введенный американским ученым Джеем Форрестером, родоначальником системной динамики. Удивительный результат стратегий, направленных на решение проблемы. Часто предполагаемое «решение» контрпродуктивно и приносит результат, обратный желаемому, поскольку проблем становится больше, прикладываются еще большие усилий, что фактически усугубляет ситуацию.

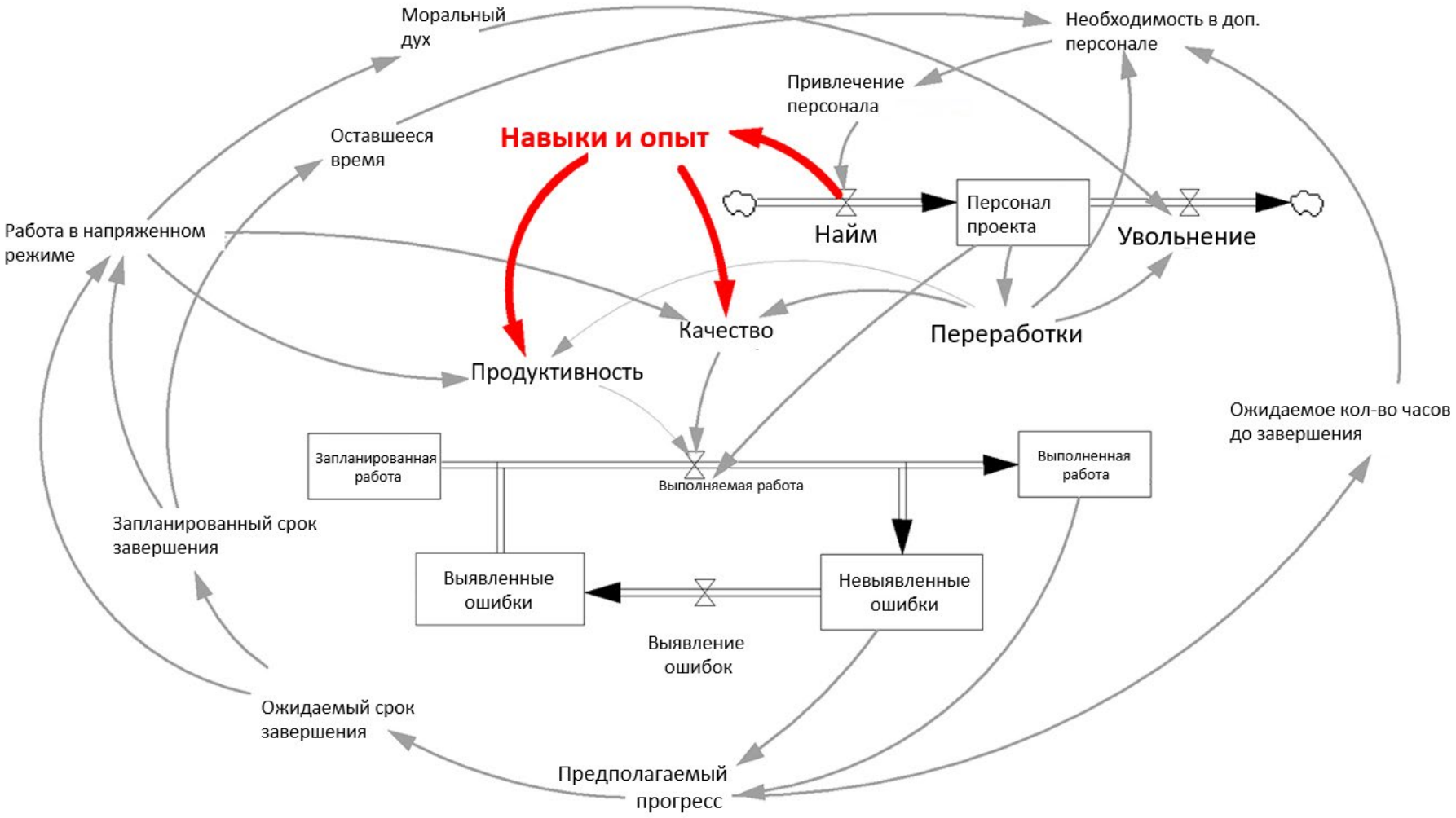


рис. 10а СНИЖЕНИЕ УРОВНЯ ПРОФЕССИОНАЛИЗМА КОМАНДЫ ИЗ-ЗА НАЙМА НОВЫХ СОТРУДНИКОВ

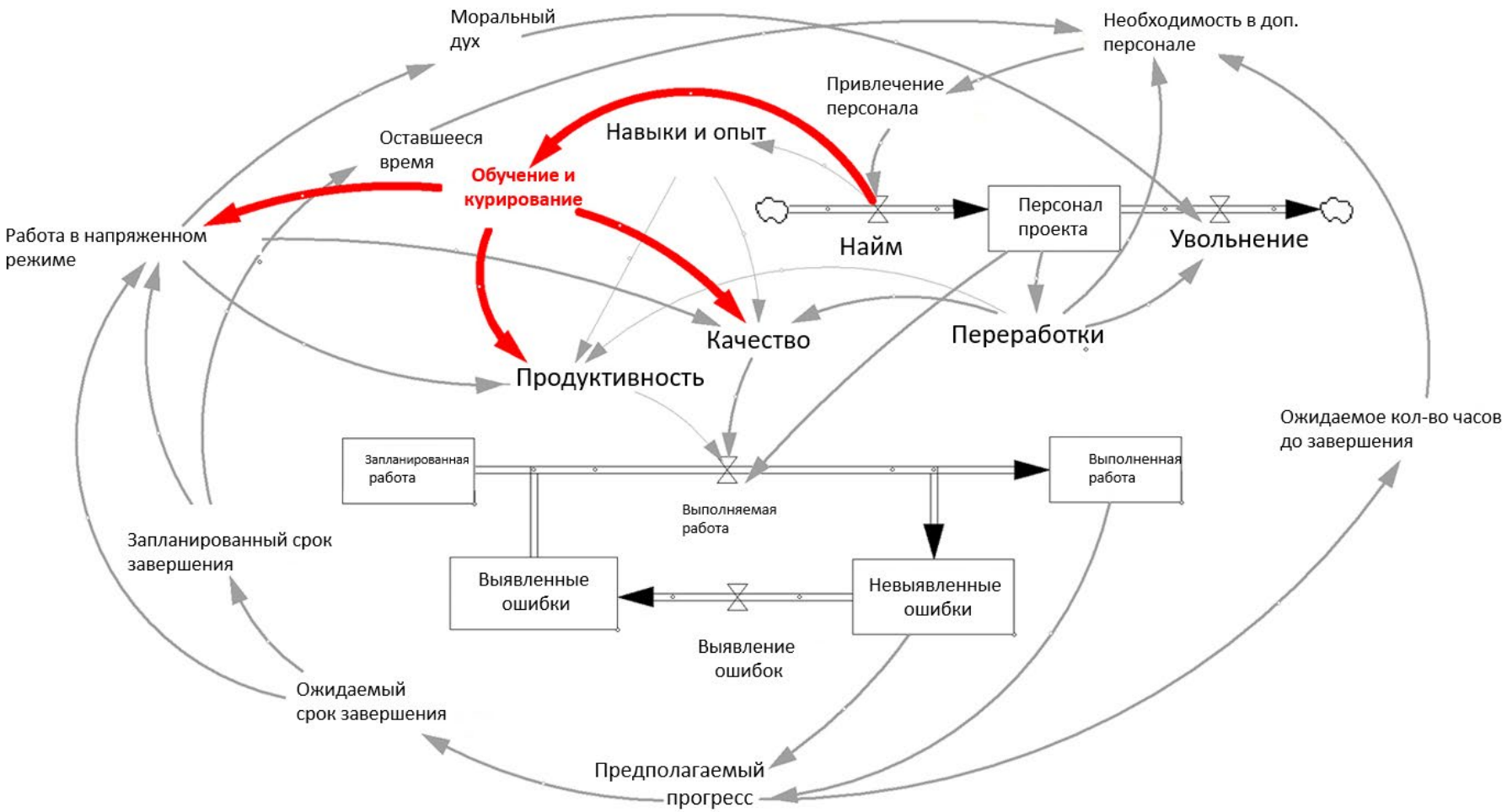


рис. 10б ОБУЧЕНИЕ И КУРИРОВАНИЕ НОВЫХ СОТРУДНИКОВ ОТНИМАЕТ ВРЕМЯ ОТ ПРОЕКТА И УСИЛИВАЕТ НАПРЯЖЕННОСТЬ РАБОТЫ

переговоры между ними и согласование вносимых изменений. В результате члены коллектива все больше морально и физически устают.

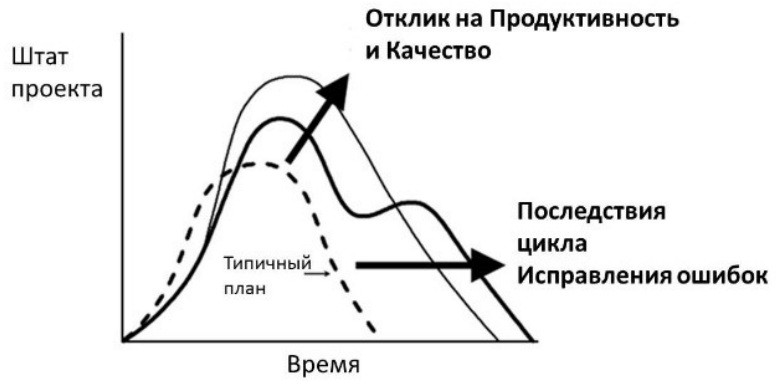


рис. 11 ВЛИЯНИЕ ТИПИЧНОГО ЦИКЛА ИСПРАВЛЕНИЯ ОШИБОК И ПРОДУКТИВНОСТИ/КАЧЕСТВА ВЫПОЛНЕНИЯ ПРОЕКТА НА КАДРОВУЮ ПОЛИТИКУ

Это способствует эмоциональному выгоранию, снижению морального духа и всевозможным вытекающим из этого негативным последствиям.

Но ситуация может продолжать ухудшаться. Проблемы, возникшие на ранних стадиях проекта, быстро увеличиваются в количестве по мере его развития – требования к работе плохо сформулированы (непонятные, неполные, слишком обобщенные), а тестирование и оценка ПО проводятся недостаточно строго и полно. Все это может вызвать необходимость последующего исправления значительного числа ошибок (NASA Office of Chief Engineer 2009: 57). Классические примеры такого развития событий наблюдаются в практике управления крупными проектами по строительству, производству, разработке новых продуктов и прочими проектами, в которых внесение изменений на этапах проектирования и конструирования влияет на стадии строительства и производства.

Иногда ошибки могут обнаружиться через месяцы и годы после того, как они возникли – это распространенная ситуация для крупных комплексных проектов (строительных, инфраструктурных, связанных с разработкой программного обеспечения и т. д.). Некоторые выявленные ошибки могут быть губительными для реализации проекта и повлечь необходимость внесения изменений, которые отбрасывают проект далеко назад.

2.2 ОПАСНЫЕ МЕСТА В РЕАЛИЗАЦИИ ТИПИЧНОГО ПРОЕКТА

Анализ комплексных проектов показал, что основными источниками рисков для реализации проектов в соответствии с установленными сроками и бюджетом, являются:

- задержки в обмене информацией или внесение изменений в первичную конструкцию;
- ресурсные ограничения (например, заторможенный запуск проекта, неполный профессиональный состав команды, вынужденная экономия средств в связи с финансовыми ограничениями и т. д.);
- новые процессы, ресурсы, сотрудники;
- организационные и управленческие изменения;
- изначально грубые допущения относительно проекта (сжатые сроки, недостаточный бюджет, неправильное восприятие общей сложности проекта).

Многие авторы, включая Боэма (Boehm, 1981), Ларсона и Вертса (Larson and Wertz,1993) и Штеклина (Stecklein et al., 2004) освещают проблему значительного роста затрат по мере осуществления проектов по разработке программного обеспечения. Ларсон и Вертс провели исследование, посвященное издержкам при запуске спутника, в котором подсчитывались издержки из-за ошибок на каждой стадии жизненного цикла проекта и были даны рекомендации для NASA по оценке последствий ошибок. В исследовании NASA было выделено четыре основных стадии (разработка технического задания, проектирование, написание кода и тестирование) проекта по разработке программного обеспечения, а также был дан прогноз, что с каждой новой стадией разработки средняя цена ошибки будет стремительно возрастать. Предположим, что цена ошибки при подготовке технического задания на проект составляет одну единицу затрат. Тогда стоимость исправления этой ошибки на стадии проектирования возрастет до 5-7 единиц, на стадии написания кода – в 10-25 раз, а на стадии тестирования – более чем в 50 раз. В исследовании NASA сравнивались разные методы подсчета роста стоимости ошибки, и было выявлено, что стоимость растет экспоненциально по мере выполнения проекта. Проекты по разработке программного обеспечения, как и многие другие крупномасштабные проекты, подвержены так называемым «ошибкам-убийцам» – ошибкам, масштабы которых велики.

Поскольку проекты (особенно комплексные) обычно представляют собой системы с тесно взаимосвязанными элементами, для них крайне важно избегать резонансного эффекта. Из-за множества взаимосвязей нелинейных петель обратной связи с нежелательными и непредсказуемыми последствиями, а также их комбинированного характера комплексные проекты часто выходят из-под контроля.

Остаточные ошибки, допущенные при разработке программного обеспечения, и небольшие отдельные огрехи часто незначительны сами по себе, но все вместе могут оказаться губительны для комплексного проекта. Эксперт NASA по разработке комплексного программного обеспечения Джеральд Холзман пришел к выводу, что добавление резервных копий и защиты от неисправностей оборачивается увеличением размера и сложности системы с «незапланированными связями между прежде не

связанными компонентами системы… В результате количество возможных ошибочных комбинаций настолько велико, что просто невозможно проверить их все в процессе систематического тестирования ПО. К примеру, всего сотня остаточных дефектов может встретиться примерно в десяти тысячах различных комбинаций» (Holzmann, 2007). Именно сочетание незначительных дефектов привело к неудачному окончанию миссии NASA «Марс Глобал Сервейер» (Holzmann, 2009).

Вывод: Комплексные проекты уязвимы перед ошибками и нарушениями. Людям, как правило, очень сложно работать с системами, компоненты которых тесно взаимосвязаны, в то время как цена ошибки подчас слишком высока (в том числе может измеряться человеческими жизнями). Способность прогнозировать разные сценарии дальнейшего развития проекта и оценивать возможные результаты управленческих решений становится крайне важным навыком. Поэтому имитационное моделирование начинает играть все более важную роль.

В следующей части главы мы подробнее поговорим о самом имитационном моделировании. Мы рассмотрим три основных подхода к проектному имитационному моделированию на примере простых моделей и обсудим плюсы и минусы каждого из них.

ОБЗОР ОСНОВНЫХ МЕТОДОВ ИМИТАЦИОННОГО МОДЕЛИРОВАНИЯ

03

ОБЗОР ОСНОВНЫХ МЕТОДОВ ИМИТАЦИОННОГО МОДЕЛИРОВАНИЯ

Среди методов имитационного моделирования, применяемых при управлении комплексными проектами, преобладает три основных системы взглядов: системная динамика, дискретно-событийное моделирование и агентное моделирование. Эти же три подхода, как правило, используются и при моделировании бизнес-систем. Рассмотрим подробнее эти подходы.

3.1 СИСТЕМНАЯ ДИНАМИКА

Системная динамика – старейший метод имитационного моделирования, восходящий к исследованиям профессора MIT (Школы менеджмента Слоуна) Джея Форрестера в 1950-х годах. Начав работать в Школе менеджмента в 1956 году, Форрестер уже был известен своими предыдущими работами. Он помог военным силам США создать первую в Штатах систему противовоздушной обороны (Semi-Automatic Ground Environment, SAGE), разрабатывал оперативное запоминающее устройство для компьютерной индустрии, а также оборудование с числовым программным управлением для промышленности. Форрестер использовал свой обширный опыт проектирования в технической сфере для создания экономических и социальных имитационных моделей.

Сначала Форрестер назвал новый метод индустриально-динамическим. Позже он переименовал метод в системно-динамический, чтобы подчеркнуть, что экономика состоит из динамически взаимосвязанных комплексных систем нелинейного и неочевидного характера.

С самого начала своего существования системно-динамический метод уделял особое внимание управленческому и организационному уровням и абстрагировался от инженерных и производственных сторон деятельности. Это было обусловлено предположением, что большая часть проблем обычно возникает именно из-за ошибок в управлении и организации.

Попробуем создать модель проекта по разработке программного обеспечения. Это хороший пример комплексного проекта, в котором задействовано много сотрудников. Системную динамику поддерживает несколько программных продуктов: VenSim, PowerSim, iThink, AnyLogic, причем все они для создания моделей используют практически единый язык задания моделей. В этом упражнении мы познакомимся с основным элементом моделирования – диаграммой потоков и накопителей.

Накопитель – это цифровое отображение моделируемых объектов, например, он может отражать сотрудников, набор требований, ряд продуктов. Любой процесс в системной динамике отображается

в виде потока, соединяющего накопители. В нашем примере разработка программного обеспечения – поток от требований к разработанному программному обеспечению (рис. 12). Таким образом, любой производственный процесс можно представить как поток от сырья к продукции.

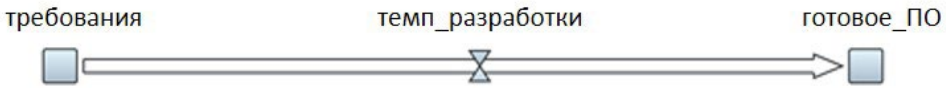


рис. 12 ПРОСТЕЙШАЯ СИСТЕМНО-ДИНАМИЧЕСКАЯ ДИАГРАММА ПРОЕКТА ПО РАЗРАБОТКЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Эта простейшая модель уже будет работать, если нам известен *темп разработки* программного обеспечения. Но он непостоянен и зависит от количества специалистов, их продуктивности и даже коммуникационных потерь при их взаимодействии. С добавлением этих параметров модель становится сложнее (рис. 13).

Эта модель наглядна и проста для восприятия, но ее нельзя использовать, пока мы не добавим числа и уравнения к связанным переменным потокам.

Продуктивность – просто константа, *персонал* – переменная, значение которой мы можем задать на этой стадии. Можно заметить, что графически отображение накопителей, потоков и переменных отличается. Это стандартный вид отображения моделей системной динамики, который

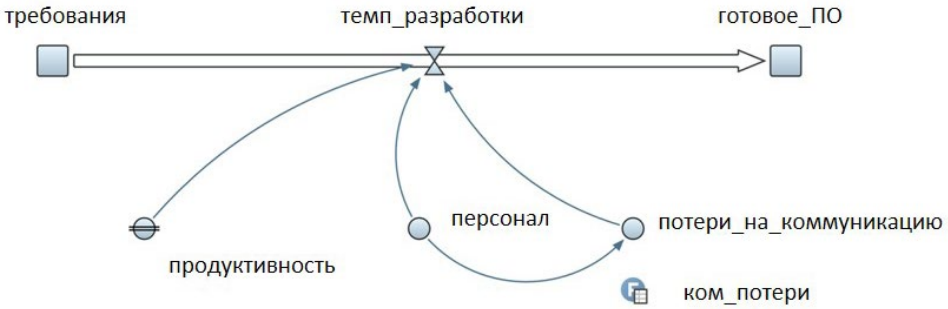


рис. 13 ДИАГРАММА ТОЙ ЖЕ МОДЕЛИ С ДОПОЛНИТЕЛЬНЫМИ ПАРАМЕТРАМИ

поддерживает ряд программных средств. Скриншоты, приведенные в этой главе, сделаны при работе в ПО AnyLogic[1]. Далее мы продолжим приводить иллюстрации моделей, созданных с помощью AnyLogic, поскольку это программное обеспечение поддерживает все основные методы имитационного моделирования. Некоторые элементы могут быть не знакомы читателю, но все значимые элементы модели будут сопровождаться пояснениями.

Потери на коммуникацию определяются таблично заданной функцией (ком_потери), что также распространено в системно-динамических моделях. Такие таблично заданные функции могут основываться на результатах наблюдений за реальной системой. Табличная функция состоит из пар аргументов и значений. Она позволяет задавать функцию, когда уравнение неизвестно. Функция будет определена, и будет выполнена аппроксимация данных.

1 - www.anylogic.ru

Горизонтальная ось на рис. 14 отражает количество сотрудников, а вертикальная –коммуникационные потери в процентах. При взаимодействии штата менее чем в пять человек коммуникационных потерь не возникает, но начиная с пяти человек и более, чем больше сотрудников занято в проекте, тем больше коммуникационные потери.

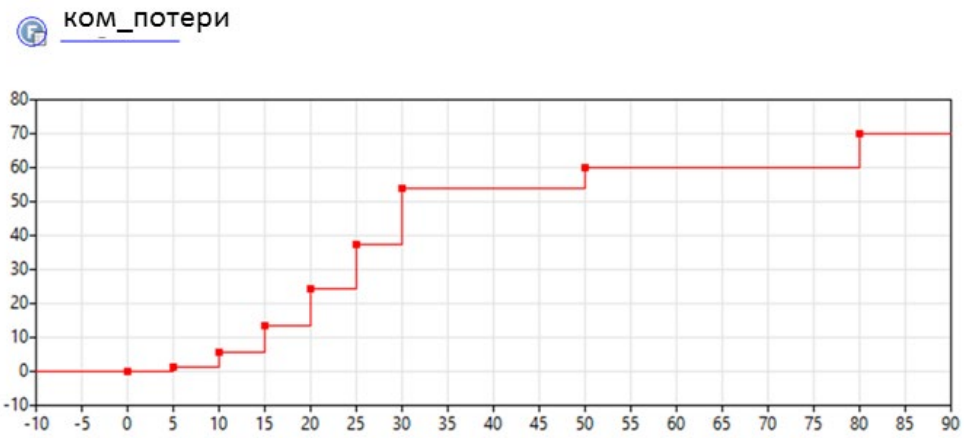


рис. 14 ТАБЛИЧНО ЗАДАННАЯ ФУНКЦИЯ, ОПРЕДЕЛЯЮЩАЯ ПОТЕРИ НА КОММУНИКАЦИЮ

В ПО AnyLogic вызов функции происходит, используя наименование этой функции с указанием параметров в скобках. Потери производительности труда из-за коммуникации будут обозначаться так:

потери_на_коммуникацию=ком_потери(персонал)

Темп разработки программного обеспечения равен произведению продуктивности, количества сотрудников и коммуникационных потерь.

$$\text{темп\_разработки} = \text{продуктивность} * (1 - \frac{\text{потери\_на\_коммуникацию}}{100}) * \text{персонал}$$

Установим параметры и запустим модель. Изначально требуется написать полмиллиона строк кода, в штате 10 сотрудников, продуктивность каждого из которых – 500 строк кода в день. В этом случае проект должен быть завершен за 106 дней (рис. 15). Этот результат понятен, поскольку каждый

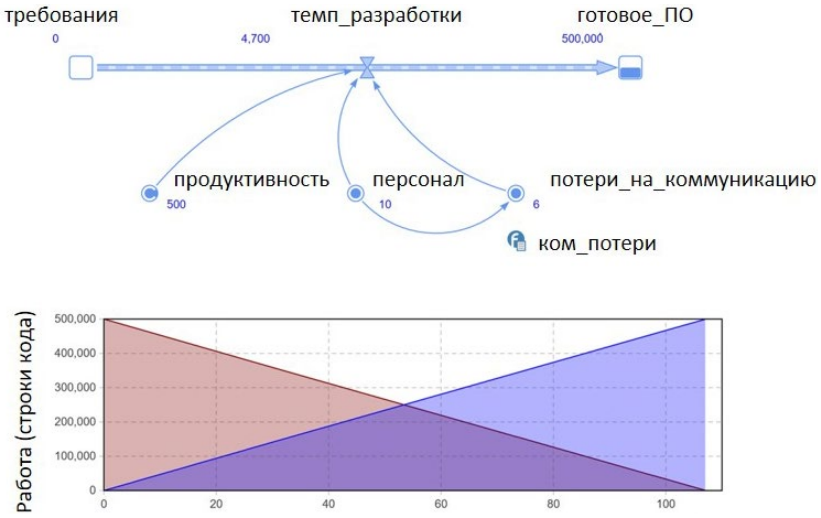


рис. 15 РЕЗУЛЬТАТЫ МОДЕЛИРОВАНИЯ ДЛЯ ШТАТА В 10 СОТРУДНИКОВ И НЕОБХОДИМОЙ ДЛИНЕ КОДА В 500 000 СТРОК

разработчик пишет по 500 строк кода в день, все вместе они пишут 5000 строк кода в день и написали бы код за 100 дней, но за счет коммуникационных потерь продуктивность снижается на 6%.

Те же результаты можно получить с помощью статического моделирования в Excel. Но чем больше деталей добавляется в модель, тем сложнее получить тот же результат с помощью электронных таблиц. Продолжим рассуждать понятиями накопителей и потоков и добавим процесс найма персонала, чтобы еще немного расширить модель.

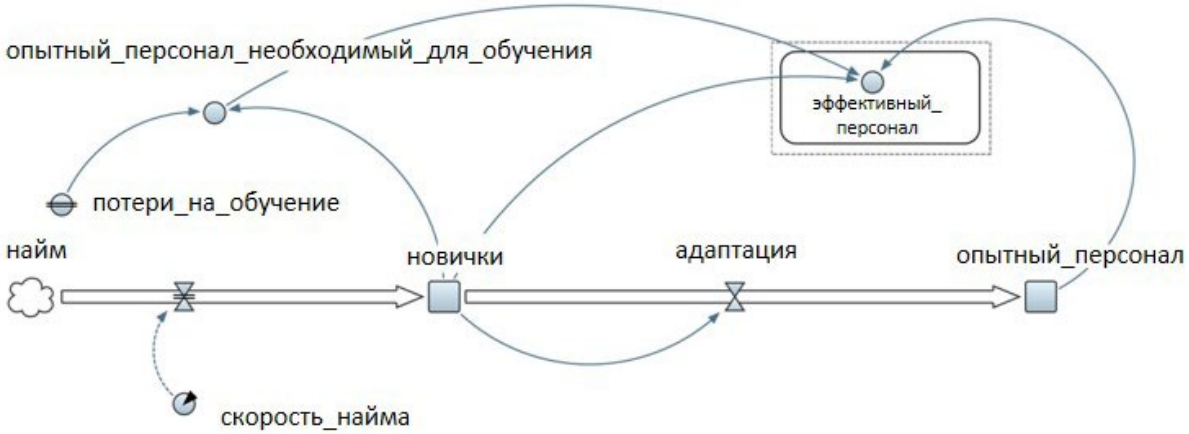


рис. 16 ПОСТУПЛЕНИЕ И АССИМИЛЯЦИЯ НОВЫХ СОТРУДНИКОВ:
ДИАГРАММА НАКОПИТЕЛЕЙ И ПОТОКОВ ВЛОЖЕННОГО КОМПОНЕНТА

Крупным достижением в развитии системной динамики в начале XXI века было распространение иерархического моделирования: создания компонентов, которые могут использоваться несколько раз на главной диаграмме модели. Это делает модель нагляднее, понятнее и удобнее в работе. Добавим в модель объект, отражающий группу разработки, и используем его как подкомпонент модели.

Структура нашей подмодели, отражающей группу разработки, приведена на рис.16 (в качестве среды моделирования используется ПО AnyLogic).

Рассмотрев диаграмму, мы выделим два накопителя – *новички* и *опытный персонал* – и два потока, отражающих *найм* в проект новых разработчиков и их *адаптацию*, под которой подразумевается знакомство с деталями проекта, его масштабом и границами.

Скорость (потока) *адаптации* сотрудников зависит от количества *новичков*, и мы должны задать его соответствующей переменной. Через месяц работы новый персонал становится опытным.

$$\text{адаптация} = \frac{\text{новички}}{\text{месяц}}$$

Опытные сотрудники занимаются обучением новичков, поэтому появляются *потери на обучение*. В нашей модели мы допустим, что для обучения четырех новичков требуется один опытный сотрудник, работающий полный рабочий день.

В интерфейсе нашей диаграммы вложенного компонента есть интерфейсная переменная *эффективный персонал*. При использовании эта переменная отображается на верхнем уровне. В нашем случае она отражает количество опытных сотрудников, которые могут работать над проектом полный рабочий день. Мы придерживаемся допущения, что продуктивность новичков составляет 80% и что лишь опытные сотрудники, не занятые в обучении, могут заниматься разработкой.

$$\begin{aligned}\text{эффективный\_персонал} \\ &= 0.8 * \text{новички} + (\text{опытный\_персонал} \\ &- \text{опытный\_персонал\_необходимый\_для\_обучения})\end{aligned}$$

Объединим эти две модели: добавим нашу подмодель на основную диаграмму, назовем ее *группа разработки* и соединим интерфейсную переменную *эффективный персонал* с переменной персонал. Теперь переменная персонал задается значением переменной *эффективной персонал,* которая вычисляется во вложенном компоненте *группа разработки.*

Предположим, что в ту же группу из 10 сотрудников в месяц будет добавляться по 3 новых разработчика. Чтобы отразить процесс найма новых сотрудников, зададим значение *показателя поступления.*

$$\text{скорость\_найма} = \frac{3.0}{\text{месяц}}$$

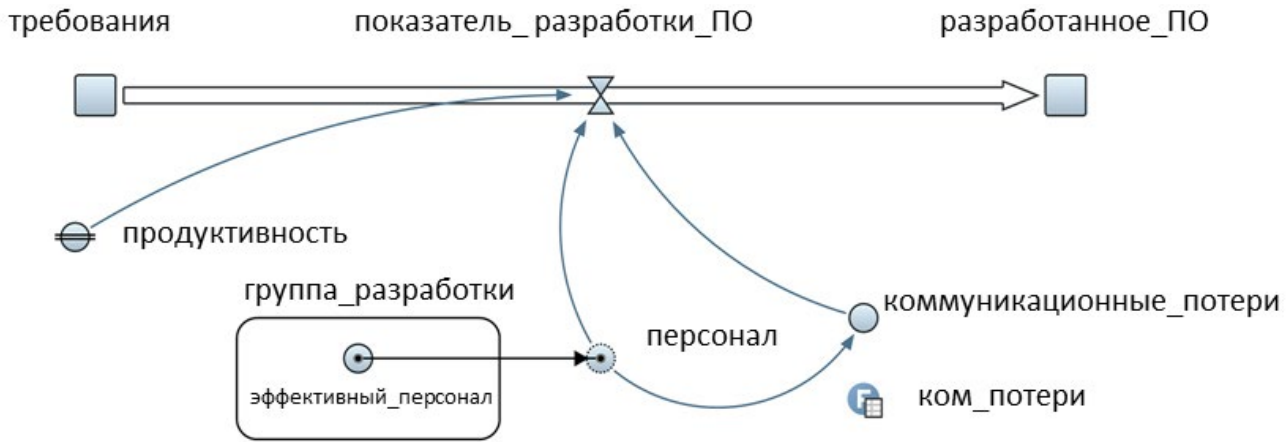


рис. 17 ПОДМОДЕЛЬ ГРУППА РАЗРАБОТКИ, ВСТРОЕННАЯ В ГЛАВНУЮ ДИАГРАММУ НАКОПИТЕЛЕЙ И ПОТОКОВ

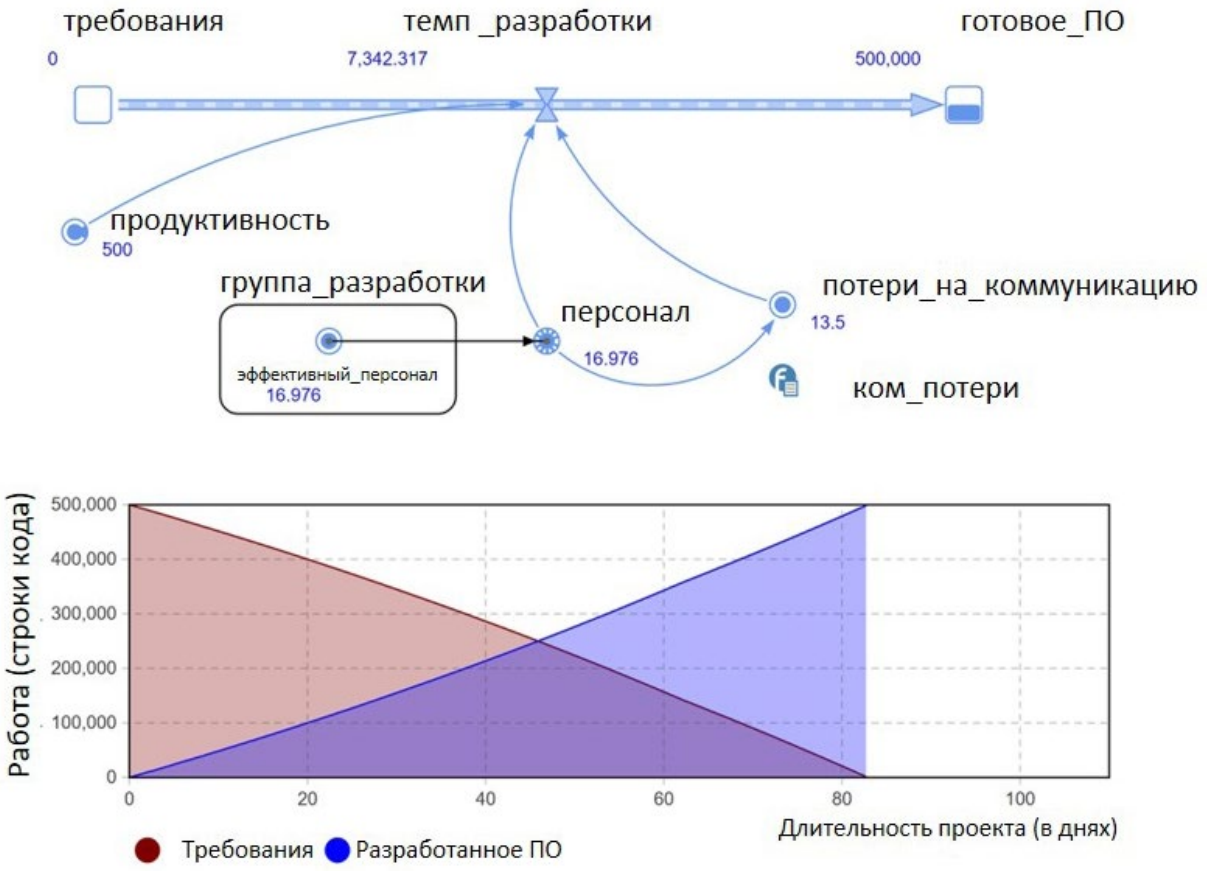


рис. 18 РЕЗУЛЬТАТЫ МОДЕЛИРОВАНИЯ, 10 СОТРУДНИКОВ, 3 НОВЫХ РАЗРАБОТЧИКА В МЕСЯЦ, 500 000 СТРОК КОДА

Если мы запустим модель, то увидим, что теперь проект выполняется за 83 дня (рис. 18).

К концу проекта у нас имеется группа из 15,432 опытных разработчиков и 2,808 новых сотрудников, продуктивность которых в совокупности составляет примерно 16,976 (рис. 19). Ситуация, когда количество сотрудников и иные неделимые величины выражены дробными числами, типична для системной динамики. Поэтому нам необходимо решить, что означает присутствие в штате 2,808 новичков. Для этой модели имеет смысл округлить это значение до 3, но для некоторых других мы просто отбросили бы дробную часть.

Теперь менеджеру проекта нужно тщательно разобраться в том, как решение привлечь новых сотрудников отразится на осуществлении проекта с точки зрения затрат и сроков. Давайте для простоты проведем еще один эксперимент с варьированием *показателя найма* от нуля до двадцати (человек в месяц) и построим график зависимости времени выполнения проекта от показателя найма. Как вы можете видеть на приведенном ниже графике, время выполнения проекта показывает нелинейную зависимость от темпа добавления новичков в проект.

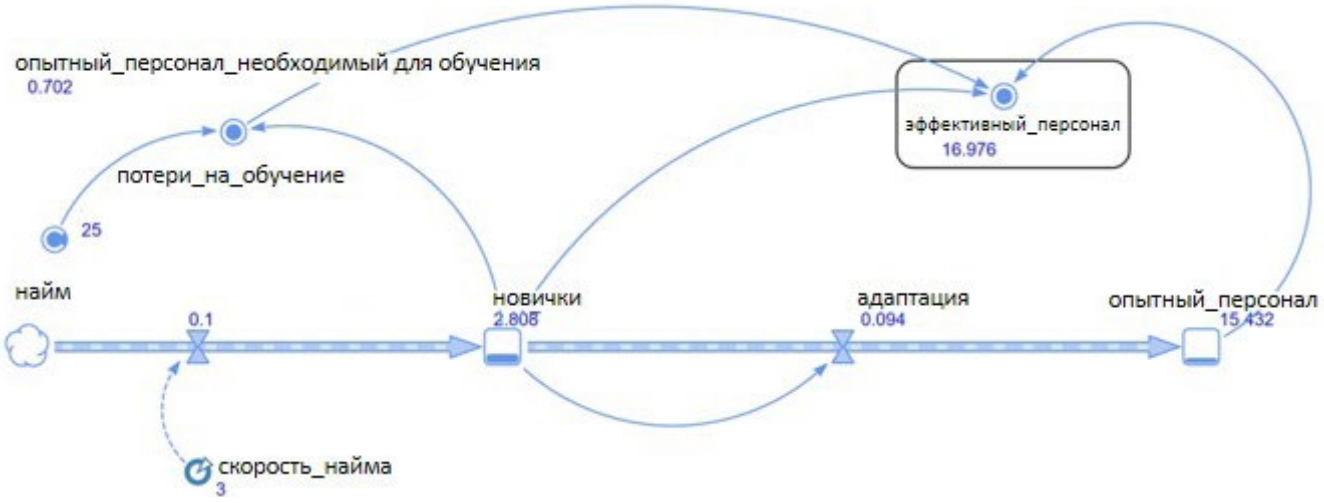


рис. 19 РЕЗУЛЬТАТЫ МОДЕЛИРОВАНИЯ, ПОДМОДЕЛЬ ГРУППА РАЗРАБОТКИ

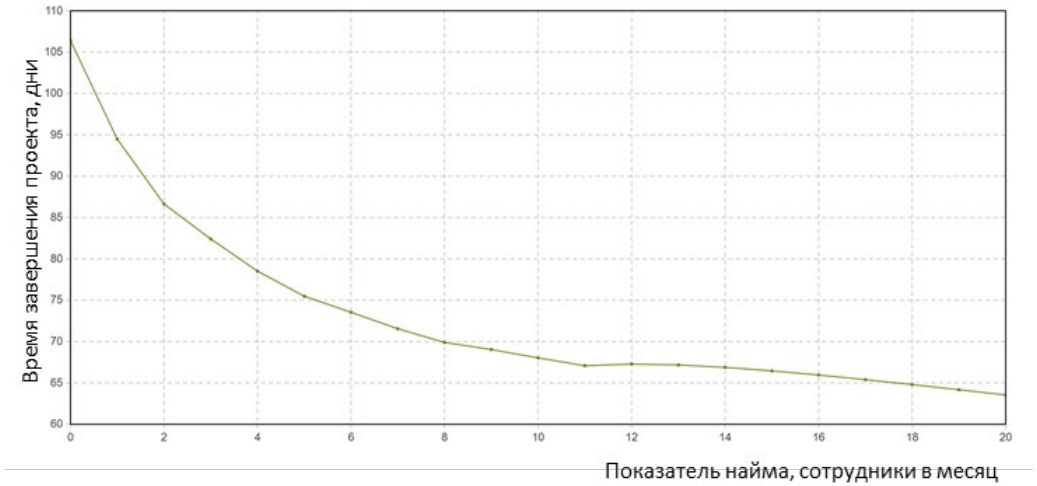


рис. 20 ВАРЬИРОВАНИЕ ПОКАЗАТЕЛЯ НАЙМА

До сих пор мы не принимали во внимание стохастику, учет которой является ключевым преимуществом имитационного моделирования. Проведем еще один эксперимент с нашей моделью, где *показатель поступления персонала* будет задаваться треугольным распределением вероятностей.

Если мы запустим модель 1000 раз и отобразим время выполнения проекта на гистограмме, то сможем проанализировать риск нарушения сроков

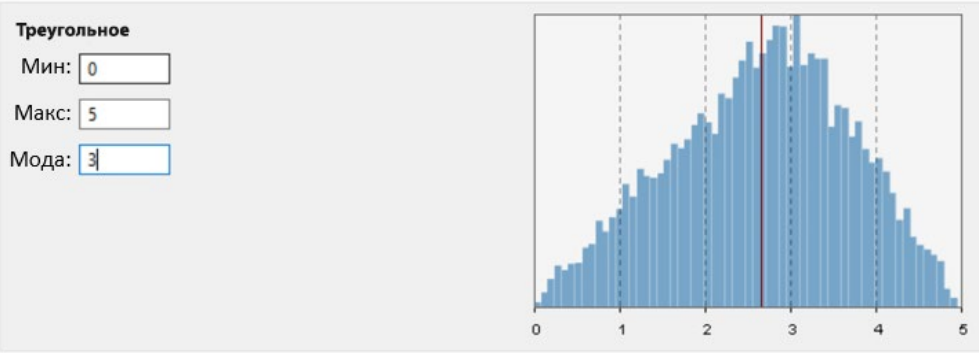


рис. 21 ТРЕУГОЛЬНОЕ РАСПРЕДЕЛЕНИЕ ВЕРОЯТНОСТЕЙ

сдачи проекта. Если наш срок – 90 дней, то из результатов моделирования нам известно, что при поступлении в нашу группу 3 новых разработчиков в месяц мы завершим проект за 83 дня, за неделю до срока. Но со случайным показателем поступления сотрудников из гистограммы мы видим, что с вероятностью в примерно 28% мы можем не уложиться в срок в 90 дней. 28% – это сумма последних двух столбцов на рисунке 22. Менеджер проекта должен задуматься о привлечении больших ресурсов, а мы – продолжить экспериментировать с моделью, чтобы помогать ему принимать решения.

Для создания этой модели мы использовали универсальное программное обеспечение для создания имитационных моделей AnyLogic. Модель находится в открытом доступе на облачном сервисе AnyLogic Cloud, так же как и эксперименты, которые мы приводили, и исходный код модели. Эти материалы можно найти по названию «Software development process dynamics» или по автору – Sergey Suslov[1].

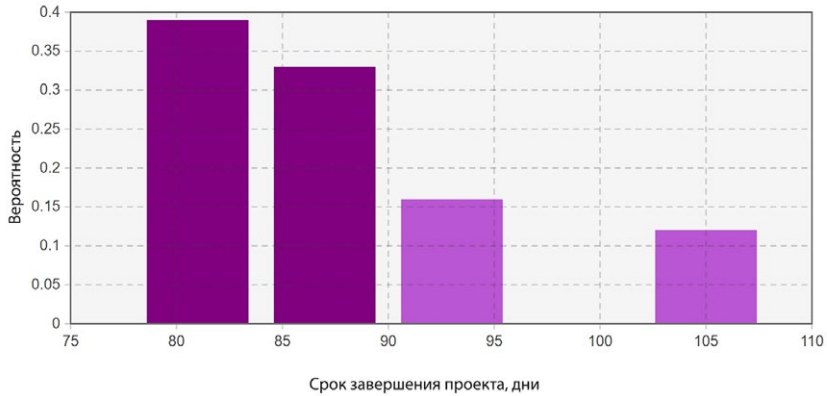


рис. 22 ГИСТОГРАММА ВРЕМЕНИ ЗАВЕРШЕНИЯ ПРОЕКТА ПРИ СТОХАСТИЧЕСКОМ ПОКАЗАТЕЛЕ НАЙМА

1 - Модель в AnyLogic Cloud

3.2 АГЕНТНОЕ МОДЕЛИРОВАНИЕ

Агентное моделирование – самый молодой из основных методов моделирования. Тем не менее некоторые агентные модели можно встретить в старых журналах и книгах, таких как публикация Томаса Шеллинга «Микромотивы и макроповедение» 1978 года выпуска (Schelling, 1978). Эти труды можно рассматривать как зарождение концепции агентного моделирования. Десятилетиями его практическое применение было ограничено уровнем развития компьютерной техники. В 90-х агентное моделирование оставалось чисто научной темой, но в 21-м веке с расцветом компьютерной техники оно стало коммерчески применимо для решения крупномасштабных бизнес-задач, более того, теперь его применяют все чаще по сравнению с другими методами.

Агентное моделирование и системная динамика – диаметрально противоположные методы. В то время как системная динамика основана на целостном описании процесса, агентное моделирование использует принцип от частного к общему, который выражается в объединенном поведении системы, складывающемся из совокупности поведения объектов (агентов). Результирующее поведение системы представляет собой совокупность действий множества отдельных агентов. Научная литература уделяет внимание различным проблемам этого метода, например, какие объекты можно моделировать с помощью агентного подхода, должны ли они обладать индивидуальным поведением, памятью, целями и т. д. Но в этой главе мы рассмотрим лишь некоторые более практические аспекты применения агентного моделирования.

В последние два десятилетия специалисты в области агентного моделирования разработали ряд полезных пакетов программных средств. По мнению авторов, самыми популярными среди них являются AnyLogic, Swarm, RePast, NetLogo. Каждое ПО обладает собственными отличительными характеристиками, сильными и слабыми сторонами.[1] Чтобы лучше понять механизм создания моделей с помощью агентов, попробуем пересмотреть концепцию нашей системно-динамической модели разработки программного обеспечения, используя агентный подход. Для разработки новых моделей мы продолжаем использовать программное обеспечение AnyLogic.

Основной объект агентной модели очень прост, поскольку включает лишь пару популяций агентов. Популяция агентов – это множество агентов, которую также называют набором или коллекцией. В нашем случае это *разработчики* и *стажеры.* Событие *найм* периодически добавляет новых *Стажеров* в систему.

рис. 23 ПОПУЛЯЦИИ АГЕНТОВ И СОБЫТИЕ

Поскольку в агентной модели не задается общее поведение системы, начнем с частного поведения, а именно поведения *разработчика.* Диаграмма состояний – это мощный инструмент для описания состояния и поведения агентов. Поведение наших агентов *Разработчика* и *Стажера* может быть графически отражено с помощью следующих диаграмм состояния.

1 - Сравнение программного обеспечения для агентного моделирования

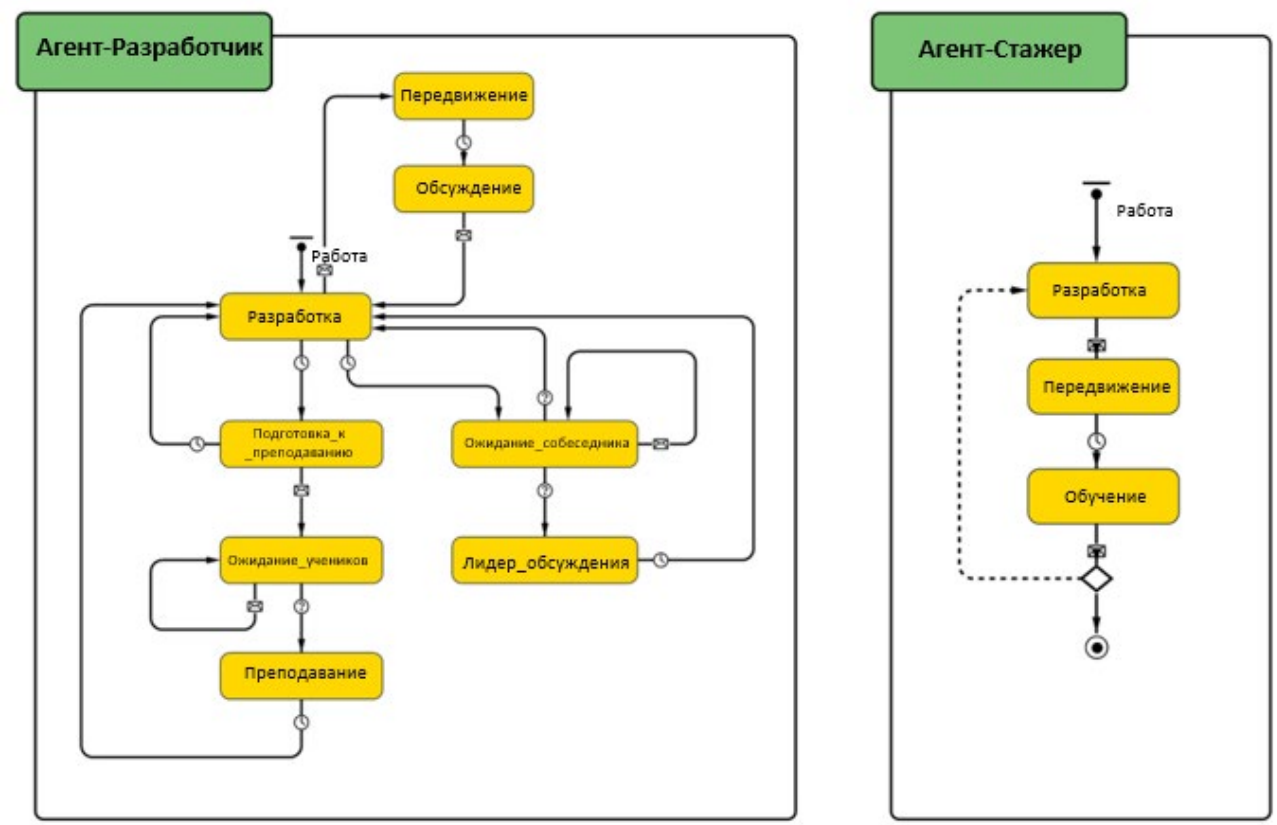


рис. 24 ДИАГРАММЫ СОСТОЯНИЙ, ОПРЕДЕЛЯЮЩИЕ ПОВЕДЕНИЕ АГЕНТОВ

Разработка – основное состояние, в котором находится агент с начала своего функционирования в рамках модели. Находясь в состоянии *Разработки,* агент производит программный продукт. *Преподавание* – это состояние, находясь в котором *Агент-разработчик* не разрабатывает код, но тратит время на обучение новых сотрудников (которых представляет *Агент-стажер*). Существует также два промежуточных состояния между *Разработкой* и *Преподаванием,* они отражают подготовку к преподаванию и ожидание учеников. Агенты общаются между собой с помощью сообщений, например, при входе в состояние *Подготовка к преподаванию* *Разработчик* посылает *Стажёрам сигнал,* сообщая, что находится в месте для обучения и готов начать преподавать. Затем в состоянии *Ожидание учеников* разработчик ждёт обратных сигналов от всех задействованных стажеров и только по достижению нужного количества переходит в состояние *Преподавание.*

Логика поведения *Стажера* гораздо проще, здесь всего три состояния: уже понятные нам *Разработка* и *Обучение,* а также *Передвижение,* которое отражает перерыв между разработкой и прохождением обучения (продолжительность *Передвижения* может различаться в зависимости от расположения офиса, в котором проходит обучение). Одна и та же модель показывает разные результаты в зависимости от пространственной среды (имеется в виду, находятся ли разработчики и стажеры в одной комнате, на одном этаже, здании, городе, стране и т. д.). Для конкретного случая мы должны задать перерыв и определить, сколько времени *Стажер* проведет в состоянии *Передвижение.* После каждого урока мы оцениваем опыт стажера и решаем, можем ли перевести его в разработчики. В этом случае агент *Стажер* удаляется, а новый *Агент-разработчик* создается.

Диаграмма состояния *Агента-разработчика* также отражает общение. Во время перерывов разработчик решает начать обсуждение и переходит в состояние *Ожидание собеседника,* при переходе посылая сообщения всем коллегам. Получив обратные сообщения от достаточного количества пришедших коллег, он становится *Лидером обсуждения,* покидая это состояние через заданный случайный перерыв. Любой *Разработчик* в состоянии *Разработки* переходит в состояние *Передвижение* по соответствующему сигналу от другого агента, который инициирует общение. Когда агент достигает *Лидера обсуждения,* он переходит в состояние *Обсуждение.*

Агентные модели обычно содержат много скриптов. В нашем случае язык Java поддерживается программным обеспечением для имитационного моделирования AnyLogic. Множество небольших операций, таких как отправка сообщений и перемещение агентов, производится за счет добавления нескольких строк кода в переходы или состояния диаграмм состояний. Например, в нашей модели мы добавляем новых стажеров, периодически выполняя 3 строчки кода.

Запустив эту модель, мы получим примерно такие же результаты, как в системно-динамической модели. Возникает вопрос, зачем же нам нужно агентное моделирование, когда существует системная динамика? Это закономерный вопрос, на который можно дать ряд ответов:

- **Создание агентной модели – это в большей степени описательный процесс.** Для получения результата мы просто описываем, как компоненты системы (т. е. агенты) действуют и общаются. Во многих случаях легче описать компоненты системы, чем определить накопители, и потоки, и особенно петли обратной связи.

- **У агентного моделирования шире область применения.** Системная динамика обычно используется для поддержки высшего уровня управления в организации, в то время как агентное моделирование гораздо лучше поддерживает решение задач среднего уровня, таких как цепи поставок, где внимание уделяется транспорту, дистанциям, конкретным уровням запаса продукции на складах и т. п.

- **Агентное моделирование – хороший инструмент для интерпретации данных компании.** Современные компании располагают огромным количеством информации, и агентные модели могут использовать эту информацию для получения неочевидных результатов. К примеру, поставщики услуг мобильной связи обладают полной информацией о своих клиентах (возраст, доход, пол и т. д.). Они могут создать модель, где каждый клиент будет представлен в виде агента, и получить точный цифровой двойник своей клиентской базы или даже сегмента рынка в соответствующем регионе.

Пример применения агентного моделирования для компании по разработке программного обеспечения[1]

Существует много хороших примеров применения агентного моделирования при управлении комплексными проектами. В частности, одна крупная компания по разработке программного обеспечения использовала агентное моделирование и описанный выше подход для проверки своей внутренней политики по управлению персоналом. Они проанализировали три альтернативные стратегии по управлению расширяющимся международным коллективом:

- Постоянное обучение и курирование.
- Замена наименее эффективных сотрудников.
- Комбинация двух описанных стратегий.

Эта компания выполняет подрядные работы, поэтому управление персоналом проекта крайне важно. Руководству необходимо постоянно решать, нужно ли расширять команду специалистов и как именно. Компания должна планировать обучение сотрудников, чтобы кроме рядового персонала, найти который достаточно просто, готовить менеджеров проектов, руководителей групп, старших разработчиков. Агентная модель помогает осуществлять такое планирование.

Изначально в большинстве агентных моделей в качестве агента выступал человек: сотрудник, клиент, пациент и т. д. С развитием технологий ситуация стремительно меняется, и теперь появляется гораздо больше моделей, в которых агентами выступают объекты, транспортные средства, склады и даже нематериальные объекты, такие как проекты. Пример использования агентного моделирования поставщиком логистических услуг (см. ниже) может помочь нам лучше понять эту область применения, поскольку модель включает таких агентов, как автомобили, площадки и т. д.

1 - Практические кейсы применения агентного моделирования

Пример применения агентного моделирования поставщиком логистических услуг

Из-за роста конкуренции в сфере логистических услуг на рынке возник спрос на программные средства и решения, которые позволили бы повысить эффективность логистических операций. Крупный оператор комплексных логистических услуг создал агентную имитационную модель доставки негабаритных грузов на дистанции в тысячи километров для проекта по строительству предприятий нефтегазовой промышленности.

Строительство уже спланировано, и среди логистических операторов проводился конкурс заявок. Для подрядчика подсчет точной стоимости доставки и планирование логистики проекта – трудоемкая задача. Причина заключается в том, что существует масса возможных вариантов осуществления проекта и необходимо принимать множество мелких решений, к примеру, какие транспортные средства лучше использовать, какой маршрут выбрать, как состыковать железнодорожный, дорожный и речной транспорт.

В агентную модель, созданную с помощью AnyLogic, включены такие агенты, как Транспорт, База (промежуточная точка, в которой можно хранить груз), Строительные площадки (пункт назначения грузов). Модель поддерживала проверку сценариев «что если», а также прогнозировала необходимое количество транспортных средств, где и когда они понадобятся.

Имитационная модель, созданная с помощью ПО AnyLogic, оказала экспертам в сфере логистики поддержку по следующим направлениям:

- выбор оптимальной конфигурации смешанных перевозок (перевозок при помощи нескольких видов транспорта) с учетом сроков поставки, стоимости транспортировки и сопутствующих рисков.
- снижение финансовых рисков и понимание вероятности выполнения установленных требований по срокам доставки.
- прогнозирование состояния транспортных средств в кратко- и среднесрочной перспективе.
- планирование бюджета и предоставление конечному клиенту более выгодных условий контракта, чем у других логистических операторов.
- оперативное осуществление финансовых расчетов для быстрого принятия решений.

Мы надеемся, что приведенные примеры из практики помогли показать разнообразие агентных моделей для управления комплексными проектами. Агентное моделирование может применяться как для концептуальных задач, характерных для верхнего уровня руководства, так и для среднего уровня обобщения, где детали проекта играют более важную роль в определении поведения системы.

3.3 ДИСКРЕТНО-СОБЫТИЙНОЕ МОДЕЛИРОВАНИЕ

Дискретно-событийное моделирование – почти такой же старый метод, как системная динамика. В октябре 1961 года инженер компании IBM Джеффри Гордон представил первую версию языка моделирования GPSS (General Purpose Simulation System – система моделирования общего назначения, первоначальное название – Gordon's Programmable Simulation System – система программного моделирования Гордона). Это считается первым случаем создания ПО для дискретно-событийного моделирования (Borshchev 2013). В настоящее время многие программные средства поддерживают этот метод. С точки зрения практического применения, мы должны отметить, что более 50% создаваемых имитационных моделей разрабатываются с применением дискретно-событийного метода или содержат компоненты, созданные с его помощью.

Основная идея дискретно-событийного моделирования заключается в том, что система рассматривается как процесс, а именно как последовательность операций, которые выполняются с разными структурными единицами (заявками). В число операций входят задержки, обеспечение различными ресурсами, выбор ветви процесса, разделение заявки, комбинирование заявок и так далее. Поскольку заявки борются за ресурсы и возможны задержки, фактически по всей дискретно-событийной модели встречаются очереди. Графически модель представляется в виде блок-схемы процесса, в которой блоки обозначают операции. Блок-схема обычно начинается с блоков-источников, которые создают заявки и включают их в процесс, и заканчивается блоками «выход из системы», которые убирают заявки из модели. Такие диаграммы привычны для деловых кругов, поскольку блок-схемы повсеместно используются для описания бизнес-процессов.

Все основные средства дискретно-событийного моделирования поддерживают похожий набор основных блоков, таких как источник, очередь, задержка/процесс/обслуживание, выход из системы. Существует более 50 программных средств, поддерживающих дискретно-событийное моделирование[1]. Некоторые из них — универсальные (Arena, AnyLogic, Simul8 и ExtendSim), а некоторые ориентированы на конкретные отрасли (Siemens Plant Simulation, FlexSim, Automod). Любое универсальное программное обеспечение позволяет использовать дискретно-событийное моделирование для решения задач, связанных с управлением проектами.

рис. 25 МОДЕЛЬ ПРОЕКТА ПО РАЗРАБОТКЕ ПО, ВЫПОЛНЕННАЯ В ВИДЕ БЛОК-СХЕМЫ ДИСКРЕТНЫХ СОБЫТИЙ

Как же создать модель проекта по разработке программного обеспечения с помощью дискретно-событийного моделирования? Прежде всего, нам понадобится блок-источник для наших требований и блок «выход из системы» для разработанного программного обеспечения. Когда у нас появились требования, нам необходимо привлечь (осуществить захват ресурса) разработчика из *опытного* персонала. Получив разработчика, мы обрабатываем часть требований и отпускаем его (Рис. 25).

Параметризуем модель конкретными данными. Исходное требование – написать 500 000 строк кода, каждая из которых является отдельной структурной единицей. Давайте введем новую переменную – *темп разработки* программного обеспечения и зададим ей следующее значение.

$$\text{темп\_разработки} = \left(\frac{1 - \text{ком\_потери}(\text{опытный\_персонал.количество})}{100}\right) * \text{продуктивность}$$

Таким образом, мы уменьшаем показатель разработки ПО из-за коммуникационных потерь точно так же, как до этого делали в системной динамике. Мы можем заново использовать таблично заданную функцию *ком_потери*.

Для времени исполнения процесса зададим значение *1/темп_разработки,* так как это время (в днях), которое одному разработчику необходимо для написания одной строки кода. У нас группа из 10 разработчиков, то есть наш процесс может производить до 10 строк кода одновременно.

1 - Исследование ПО для имитационного моделирования, ORMS Today

Запустив модель, мы получим тот же результат, что прежде получали с помощью системно-динамической модели: время завершения проекта – 106 дней.

Рассмотрим, как отразить в модели поступление новых разработчиков. Можно начать с добавления новой, изначально пустой группы ресурсов под названием *новички*. Эту группу мы используем, когда осуществляем *захват разработчика*.

Скриптовые языки программирования – «необходимое зло» при создании модели на профессиональном уровне. Для одних пользователей это любимая часть моделирования, а другие испытывают из-за этого больше всего трудностей. Возможность написания скриптов делает инструменты моделирования очень мощными и гибкими, но это усложняет создание

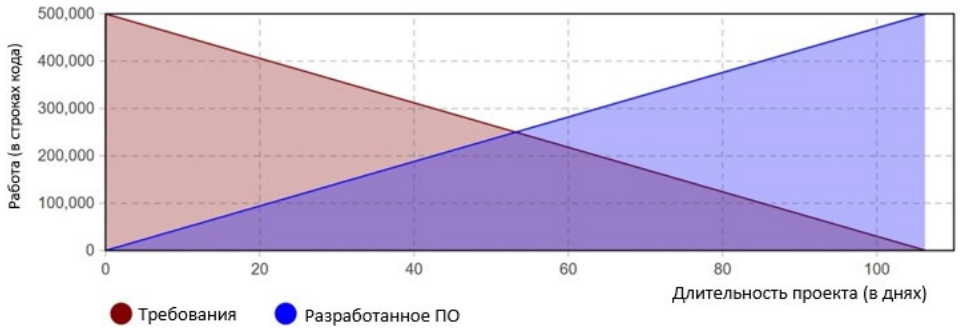


рис. 26 РЕЗУЛЬТАТЫ ДИСКРЕТНО-СОБЫТИЙНОГО МОДЕЛИРОВАНИЯ, 500 000 СТРОК КОДА, 10 ЧЕЛОВЕК

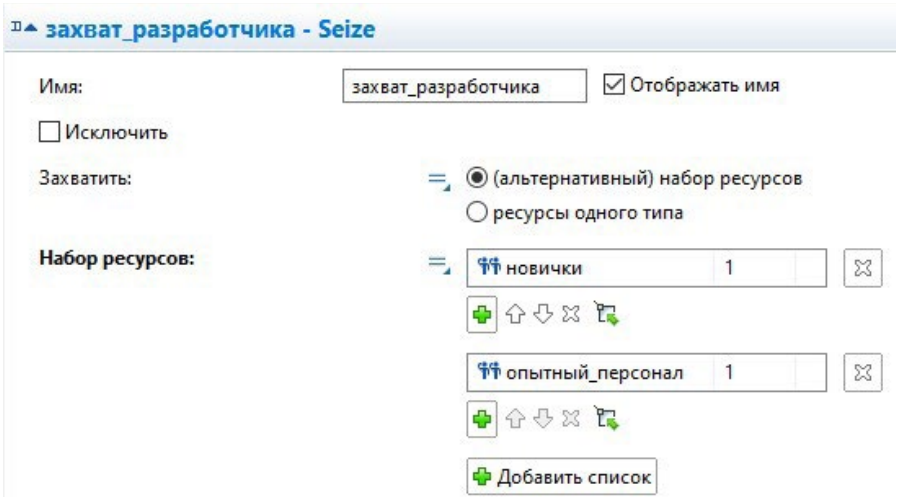


рис. 27 НАСТРОЙКА БЛОКА ЗАХВАТ РАЗРАБОТЧИКА В ПО ANYLOGIC

моделей и работу с ними. В нашей модели мы используем выражение, написанное на языке Java, чтобы задать *темп_разработки,* уникальный для каждой строки кода, которая производится в блоке процесса. Это выражение запускается каждый раз, когда мы берем разработчика.

Давайте модифицируем код на языке Java, который мы выполняем, когда осуществляем *захват разработчика:*

```
темп_разработки=
продуктивность*
(1-ком_потери(опыт_персонал.size()+новички.
size())/100)*
(((Разработчик)unit).опытный ? 1 : 0.8)*
```

```
(1-потери_на_обучение/100* новички.size()/опыт_
персонал.size());
```

Темпу разработки присваивается новое значение, равное продуктивности, уменьшенной за счет трех множителей:

- Первый отражает потери, связанные с коммуникацией.
- Второй изменяет продуктивность в случае, если разработчик является новым сотрудником. Продуктивность опытного сотрудника — 100%, новичка – 80%.
- Последний множитель отражает, что мы должны перераспределить время опытных сотрудников на преподавание. Здесь мы перемножаем *потери на обучение* на долю *новичков* от *опытного персонала.*

Это выражение отражает влияние нового персонала на коммуникационные потери, связанные с обучением, и на продуктивность из-за меньшей опытности нового сотрудника. Скорость производства для каждой заявки, отражающей строку кода, будет уникальна в зависимости от разработчика, текущего размера группы и ее структуры.

Последняя и наиболее важная часть – процесс поступления новых разработчиков. Как наиболее важная часть процесса, она начинается с блока-источника и заканчивается блоком *окончание_обучения* (Рис. 28). Новые разработчики покидают блок *поступление персонала* согласно *показателю найма* персонала в месяц. Когда появляется новый разработчик, мы увеличиваем количество ресурсов в группы *новички.* После этого новые сотрудники поступают в блок обслуживания *обучение*, где сразу же приступают к обучению сроком в месяц. По окончании обучения мы увеличиваем *опытный_персонал*, добавив к нему новый ресурс, и уменьшаем количество заявок-ресурсов в группе *новички.*

Когда мы запускаем модель с теми же параметрами, мы получаем немного отличающиеся результаты, что связано с дискретностью процесса поступления персонала. Время выполнения проекта составляет 88 дней, и к его окончанию у нас имеется 15 опытных разработчиков и 3 новых сотрудника.

Модель и ее исходный код выложены в облаке AnyLogic Cloud[1]. Ее можно найти по наименованию «Software Development Process Dynamics DE» или по имени автора – Sergey Suslov.

Вы можете воспользоваться бесплатной версией AnyLogic PLE[2], чтобы изучить внутренние элементы модели или даже продолжить ее разработку.

Дискретно-событийное моделирование обычно используется для технической составляющей управления проектами, когда менеджеры и

рис. 28 РЕЗУЛЬТАТЫ ДИСКРЕТНО-СОБЫТИЙНОГО МОДЕЛИРОВАНИЯ, 500 СТРОК КОДА,10 ЧЕЛОВЕК

1 - Модель

2 - Скачать AnyLogic PLE

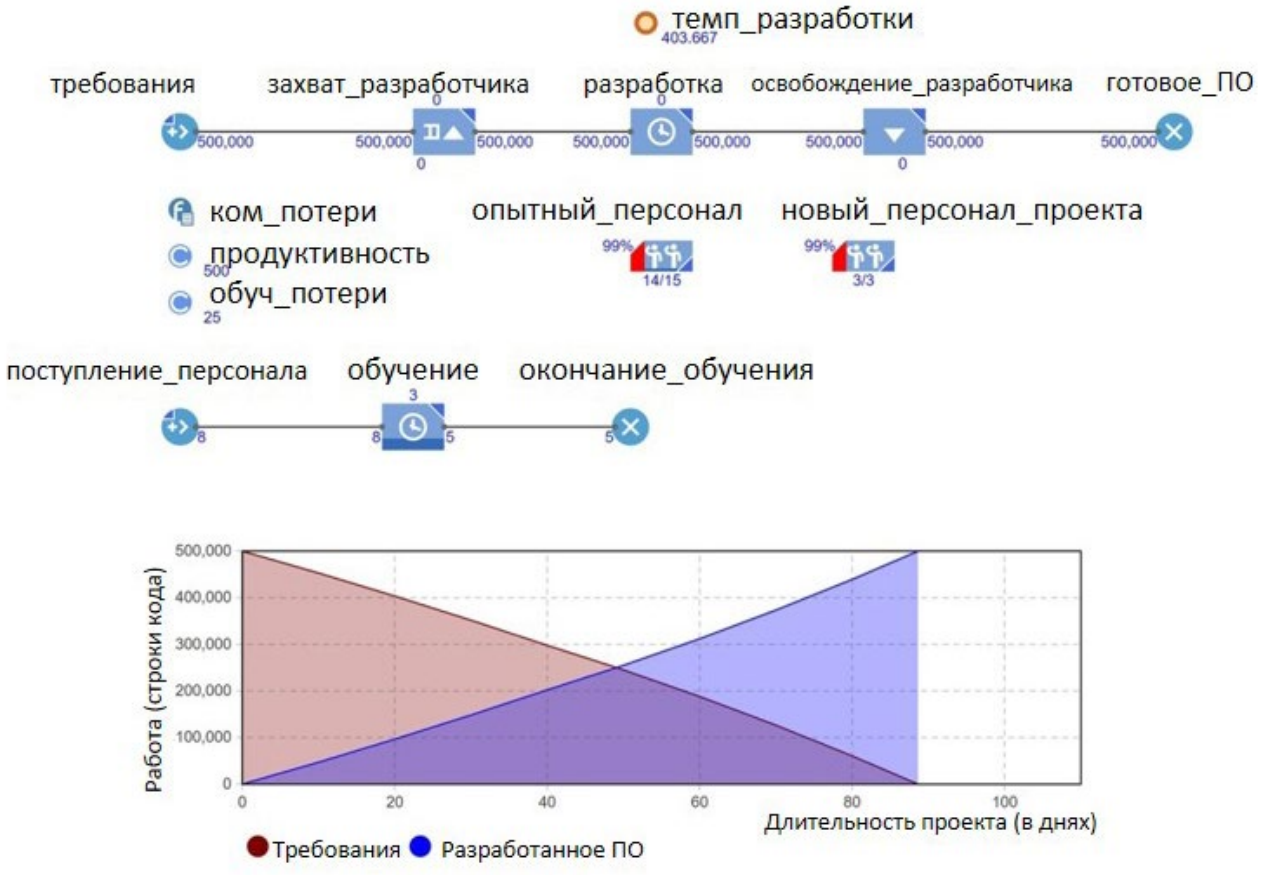


рис. 29 РЕЗУЛЬТАТЫ ДИСКРЕТНО-СОБЫТИЙНОГО МОДЕЛИРОВАНИЯ, 500 СТРОК КОДА, 10 СОТРУДНИКОВ, 3 НОВЫХ СОТРУДНИКА В МЕСЯЦ

инженеры пытаются предсказать время выполнения каждого этапа проекта и снизить риск нарушения сроков по техническим причинам. По сравнению с системной динамикой и агентным моделированием дискретно-событийное моделирование используется для низкого и среднего уровня абстракции, когда детали технологического или бизнес-процесса играют ключевую роль. Продемонстрируем на примере, как эксплуатационные ограничения влияют на возможность достижения целей проекта.

Пример применения ИМ компанией Consolidated Contractors Company

Разберем пример из практики компании Consolidated Contractors Company (CCC). Это крупнейшая строительная компания на Ближнем Востоке и 18-ая в мире. У CCC есть офисы в более чем 40 странах мира и штат из 130 тыс. сотрудников. В списке успешно выполненных компанией проектов – нефтегазовые заводы, трубопроводы, энергетические и опреснительные установки, предприятия легкой промышленности, водопроводно-канализационные очистительные сооружения, аэропорты и морские порты, предприятия инженерно-строительного обеспечения, плотины, водохранилища и водораспределительные системы, дорожные системы и небоскребы. Сразу после подписания

контракта на подготовку площадки (земляные работы с перемещением примерно четырех миллионов кубометров грунта), клиент и местные органы власти выдвинули новые,

более жесткие требования к проведению работы. К этим требованиям относилось следующее:

- Грузовикам теперь разрешалось передвигаться по территории промплощадки со скоростью 10 км/ч вместо изначальных 20 км/ч.
- Грузовикам предписывалось следовать до пункта выгрузки и обратно по конкретному маршруту с большим количеством светофоров, перекрестков, круговых развязок и постов охраны. Таким образом, средняя скорость на маршруте до пункта выгрузки и обратно в 40 км/ч, которая закладывалась изначально, не могла быть соблюдена.
- Максимальные разрешенные объем и загрузка грузовиков были снижены с 32 до 15 кубических метров.
- Максимальное число поездок грузовиков было ограничено до 100 в час.
- Лишь один из четырех изначально намеченных доступов на площадку был одобрен после подписания контракта, что привязало весь транспорт на площадке к одному выезду.
- Для выгрузки грунта на пункте выгрузки теперь была разрешена лишь одинарная десятичасовая смена, вместо прежде назначенной сдвоенной двадцатичасовой смены.

Принятые ограничения означали, что план выполнения проекта подвергнется серьезным изменениям, как и финальная стоимость его реализации. Количество и сложность ограничений не позволяли вручную оценить их влияние на время выполнения поставленных технических задач. Поэтому компания обратилась с просьбой к группе специалистов в области имитационного моделирования, чтобы те помогли оценить влияние принятых ограничений и обосновать запрос компании ССС на дополнительное время. Компания ССС уже обладала хорошо зарекомендовавшей себя имитационной моделью строительных проектов, выполненной с помощью ПО AnyLogic. Эта модель использовалась для базовой оптимизации строительства, определения необходимого оборудования и времени для проведения земляных работ. Моделирование земляных работ требует ввода большого количества исходных параметров, включая предполагаемую среднюю скорость, с которой грузовики будут двигаться по своему маршруту. С новым маршрутом грузовиков, навязанным проекту, было крайне сложно рассчитать среднюю скорость, с которой будут двигаться грузовики.

Поэтому еще более сложная имитационная модель была разработана в ПО AnyLogic, чтобы воспроизвести прохождение грузовиками (как загруженными, так и пустыми) отдельных отрезков маршрута. Использование технологий оптимизации AnyLogic было полезно для проведения экспериментов в безрисковой среде и эффективного управления работами в этом проекте. В имитационной модели для каждого отрезка маршрута использовалось стохастическое распределение времени его прохождения. Затем имитационную модель запустили таким образом,

чтобы грузовики проехали маршрут туда и обратно 10 000 раз. На основании полученных данных была вычислена средняя скорость на каждом маршруте с учетом степени загрузки.

Зачем использовать имитационное моделирование в строительстве? Выбор дискретно-событийного имитационного моделирования в AnyLogic был обусловлен тем, что это программное обеспечение позволило компании ССС:

- Очень быстро разработать модель, отражающую маршрут и его отрезки.
- Добавить карту маршрутов и наложить на нее анимацию прохождения их грузовиками, чтобы визуализация упростила объяснение деталей работы лицам, принимающим решения.

Средняя скорость грузовиков, полученная благодаря имитационной модели прохождения маршрутов грузовиками, затем была загружена в имитационную модель земляных работ наряду с остальными новыми ограничениями. Они включали:

- Объем загрузки грузовиков и количество рабочих смен (рабочих часов) в день, которые позволили создать множество сценариев, отражающих изначальные прогнозные требования ко времени и оборудованию, необходимым для выполнения всех операций.
- Прогноз текущих потребностей во времени и оборудовании, необходимых для выполнения всех операций (расчет влияния ограничений на потребности во времени и технике).
- Сценарий, позволяющий смягчить воздействие новых требований.

Таким образом, моделирование было осуществлено в два этапа: была использована модель, отражающая прохождение грузовиками заданного маршрута, и результаты работы этой модели были загружены в имитационную модель земляных работ. Это позволило компании ССС очень быстро подсчитать влияние новых более строгих ограничений и найти сценарии, при которых это влияние было бы минимальным, а также использовать полученные данные в переговорах с клиентом об увеличении сроков реализации проекта. Обе имитационные модели были выполнены в ПО AnyLogic, поэтому обладали достаточной гибкостью для учета всех новых ограничений. Дискретно-событийное имитационное моделирование зарекомендовало себя как хороший вспомогательный инструмент для управления строительными проектами. Результаты, полученные с помощью имитационной модели строительства, убедили клиента увеличить общий срок выполнения земляных работ на 50% и позволить компании работать в две смены. Это позволилоф снизить дополнительные затраты на 18% от изначальной общей стоимости контракта.

3.4 МНОГОПОДХОДНОЕ ИМИТАЦИОННОЕ МОДЕЛИРОВАНИЕ: ОСНОВНЫЕ ПЛЮСЫ И МИНУСЫ

Несмотря на то, что теоретически мы можем создать модель одного и того же проекта по разработке ПО тремя разными методами моделирования, на практике так поступают редко. Метод моделирования часто можно подобрать, следуя советам, приведённым далее:

- Если у вас много отдельных объектов, используйте агентное моделирование.
- Если имеется только информация об общих взаимосвязях, используйте системную динамику.
- Если систему легко описать как процесс, используйте дискретно-событийный подход[1].

В реальности большинство случаев – комплексные, поэтому удобнее отображать разные части системы с помощью разных методов. При использовании только одного метода, ваши возможности в отражении всех взаимосвязей и сложностей реальных производственных или бизнес-систем сильно ограничены. Некоторые элементы системы придется исключить или разрабатывать обходные варианты. В таких случаях имеет смысл рассмотреть возможность совмещения в одной модели различных методов моделирования, это называется многоподходным моделированием.

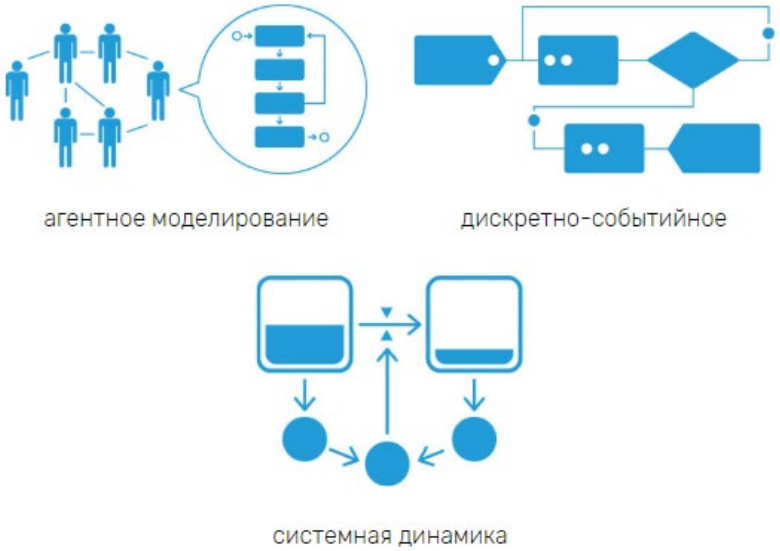


рис. 30 МЕТОДЫ ИМИТАЦИОННОГО МОДЕЛИРОВАНИЯ

Большинство современных вендоров, занимающихся имитационными моделями, начали внедрять хотя бы некоторые элементы многоподходного моделирования. Есть основания предполагать, что постепенно именно многоподходное моделирование станет отраслевым стандартом для универсальных программных средств имитационного моделирования. В книге «The Big Book of Simulation Modeling» (Borshchev, 2013) Андрей Борщев предложил ряд типичных конфигураций многоподходных моделей:

- **Агенты взаимодействуют с процессной моделью.** К примеру, оборудование, заданное с помощью агентов, влияет на процесс работы конвейерной системы.

1 - Методы имитационного моделирования

рис. 31 КОМПЛЕКСНАЯ МОДЕЛЬ ЦЕПИ ПОСТАВОК. ЗДЕСЬ ВЫ МОЖЕТЕ НАБЛЮДАТЬ, КАК ПРОИЗВОДСТВО, ДИСТРИБУЦИЮ И РЫНОК МОЖНО ОБЪЕДИНИТЬ В ОДНОЙ МОДЕЛИ С ПРИМЕНЕНИЕМ РАЗЛИЧНЫХ ТЕХНИК. ДИСКРЕТНО-СОБЫТИЙНАЯ МОДЕЛЬ ОПИСЫВАЕТ ПРОЦЕССЫ В РАМКАХ КАЖДОГО СКЛАДА. ЗАТЕМ СКЛАДЫ ПРЕДСТАВЛЯЮТСЯ В КАЧЕСТВЕ АГЕНТОВ СЕТИ ПОСТАВОК. НАКОНЕЦ, РЫНОК, КОТОРЫЙ УПРАВЛЯЕТ СИСТЕМОЙ, МОДЕЛИРУЕТСЯ СИСТЕМНОЙ ДИНАМИКОЙ.

- **Процессная модель, интегрированная с системно-динамической моделью.** Блок-схема бизнес-процессов может изменять переменные системно-динамической диаграммы для отчета о прибылях и убытках компании.

- **Системная динамика внутри агентов.** Производственные процессы на фабриках могут быть отражены с помощью системной динамики и одновременно использоваться в качестве агентов в модели цепи поставок.

- **Процесс внутри агентов.** Хорошим примером является система складов на рис. 31.

- **Агенты временно выступают единицами процесса.** К примеру, в модели регионального здравоохранения, где люди отражены с помощью агентов, которые попадают в больницу и проходят через блок-схему как дискретно-событийные заявки.

Мы рекомендуем вам руководствоваться техническими рекомендациями, приведенными в книге Андрея Борщева, при решении задач управления проектами на этапе концептуализации.

ЗАЧЕМ НУЖНЫ ИМИТАЦИОННЫЕ МОДЕЛИ?

ВЫВОДЫ

ВЫВОДЫ. ЗАЧЕМ НУЖНЫ ИМИТАЦИОННЫЕ МОДЕЛИ?

Имитационное моделирование позволяет рассмотреть проект системно. Общая сложность даже не очень больших проектов, как правило, быстро превосходит человеческие возможности по рациональной оценке динамики системы. Лауреат Нобелевской премии Герберт Саймон ввел понятие «ограниченной рациональности», чтобы подчеркнуть этот феномен (Simon 1972, 1979; Dörner 1996). Имитационные модели позволяют нам выявить, казалось бы, парадоксальные факты и сделать из них правильные выводы. К примеру, Ли (Lee et al., 2007) провел исследование и обнаружил, что вопреки общепринятому мнению о том, что для минимизации задержек необходимо все операции осуществлять в кратчайшие сроки, на самом деле в разумных масштабах в проектах допускаются задержки, которые неизбежны, но не обязательно вредят выполнению проекта в целом. Опытные менеджеры проектов интуитивно предполагают, что небольшие задержки в проекте допустимы.

Имитационные модели являются подходящими инструментами для проведения подобных системных расчетов и сравнения различных сценариев. Имитационное моделирование может учитывать много субъективных факторов, которыми часто пренебрегают в традиционных подходах, например, моральный дух коллектива, продуктивность, мотивация, усталость, ухудшение качества и т. д. С годами динамическое моделирование обрело свой язык и методологию для выражения различных материальных и нематериальных переменных (факторов), которые могут быть измерены (можно провести численные подсчеты) и добавлены в модель.

Ряд исследований описывает успешное применение имитационного моделирования для поддержки самых разных проектов: по сооружению автомобильных дорог (Ford et al., 2004), строительству (Pena-Mora and Park, 2001; Love et al., 2002), разработке полупроводников (Ford and Sterman, 1998), производству муки (Godlewski et al., 2012), разработке ПО (Abdel-Hamid and Madnick, 1982), военных (Lyneis et al., 2001) и кораблестроительных проектов (Cooper, 1980) и др.

Достижения в развитии ПО для имитационного моделирования сделали разработку моделей проще и быстрее, позволяя подключать к процессу менеджеров. Профессор Массачусетского технологического института Джон Штерман объясняет это так:

> Программные инструменты моделирования традиционно были чем-то сложным и недоступным для всех, кроме специально обученных аналитиков. Как следствие, модели разрабатывались сугубо экспертами… Проблема заключалась в том, что менеджеры (которые предположительно должны были пользоваться результатами моделирования) не могли участвовать в разработке модели. Если разработчики создавали простую модель, их критиковали за пренебрежение важными связями. Если они

> создавали сложную модель, то их критиковали за то, что никто не может в ней разобраться… Решение этой проблемы заключается в активном привлечении менеджеров к процессу создания модели. Современные инструменты моделирования дают им возможность полноправно участвовать в разработке модели. Отточенные интерфейсы и интуитивно понятный дизайн позволяют менеджерам и сотрудникам всего предприятия использовать, тестировать и изменять сложные модели.
>
> *Sterman, 1992: 11–12*

Метод групповой разработки модели был создан и развит Ричардсоном и Андерсоном (1995, 2010), Венниксом (1996), Лейном (2000), Андерсеном и соавторами (2007) и другими. Изначально это были 1-2-дневные рабочие встречи, на которых опытные эксперты в области имитационного моделирования и высшее руководство компании-клиента обсуждали концептуальную структуру системы, после чего уже создавалась модель. В последнее время эта практика эволюционировала в так называемое «коллективное системное моделирование» (Van den Belt, 2004; Stave, 2010 и др.), «…которое объединяет групповую разработку модели и опосредованное моделирование, но делает акцент на постоянной совместной работе над моделью на протяжении всего времени разработки, от неучастия до высокого уровня вовлеченности» (Hovmand, 2014: 17).

В рамках групповой работы участники могут совместно создавать очень разные модели. Это может быть как простая карта, отражающая общую суть проекта, так и сложная компьютерная имитационная модель. Даже простые карты, которые используются на ранних стадиях процесса моделирования, хорошо зарекомендовали себя в постановке задач и получении полезных сведений о ключевой структуре обратных связей моделируемого проекта. Модели, созданные методом групповой разработки, отличаются более высоким качеством благодаря совместной работе опытных практиков имитационного моделирования и отраслевых специалистов с глубокими экспертными знаниями по своей специальности. К тому же групповая разработка моделей с ранним привлечением клиентов или других ключевых заинтересованных лиц позволяет завоевать их доверие и на более поздних стадиях разработки мотивировать конечных пользователей применять модель на практике.

Готовая имитационная модель может применяться на разных этапах выполнения проекта рядом способов. К примеру, Линейс (Lyneis et al., 2001) предлагает использовать имитационные модели менеджерам проекта:

- **на предварительном этапе** для конкурса заявок или составления календарного плана, оценки рисков и анализа возможностей конкурентов (чтобы при наличии необходимой информации из открытых источников оценить, во сколько им обошелся бы рассматриваемый проект);
- **во время выполнения проекта** для управления рисками, управления изменениями (анализ возможного влияния вносимых заказчиком изменений на стоимость и сроки выполнения проекта) и оценки изменений процесса (общее влияние изменений для расчета соотношения затрат в ближайшей перспективе и долгосрочных выгод);
- **по завершении проекта** для сопоставительного анализа контрольных характеристик, обучения и развития персонала (как тренажер для руководства, который может передать опыт и уроки выполненного проекта).

Рекомендации для менеджеров проектов :

- Может быть полезно взглянуть на управление проектами не как на последовательность отдельных задач, а как на потоки работы, большая часть которых состоит из цикла исправления ошибок. Важно наладить процесс тщательного и своевременного выявления ошибок, чтобы обнаруживать ошибки тогда, когда их можно исправить быстро и с наименьшими усилиями. Раннее решение проблемы значительно сокращает потери.

- Менеджеры могут влиять на выполнение проекта, но они должны принимать во внимание сложный и неочевидный характер задействованных петель обратной связи. Избегайте общепринятых суждений (например, «Давайте наймем больше людей») и примените системный подход.

- Опасайтесь самоподкрепляющихся петель обратной связи (переработки, стремительное пополнение штата, работа в напряженном режиме и т.д), которые быстро выводят проект из-под контроля. Для сохранения контроля над издержками проекта важно осознавать причины появления этих порочных замкнутых кругов.

- Обычно выходит гораздо дешевле немного сдвинуть сроки (если есть такая возможность), чем любыми силами закончить проект вовремя.

- Большее количество изменений на более поздних стадиях создает больше трудностей, причем в непропорциональном размере.

- Косвенные издержки сбоев при осуществлении проектов гораздо более опасны и обычно ведут к потерям в десятки раз выше, чем прямые издержки, связанные с перерасходами.

- Постарайтесь заложить в бюджет перерасход в 10-50%, чтобы иметь возможность оперативно решать возникающие на ранних стадиях проблемы – крупные проблемы проекта могут легко привести к издержкам, в разы превышающим изначальный бюджет проекта.

- Старайтесь создавать имитационные модели и работать с ними: теория и практические советы у вас уже есть.

Имитационное моделирование – это мощный инструмент управления проектами. С развитием специализированных программных средств и распространением знаний о теории и практике моделирования создание имитационных моделей комплексных проектов становится проще и дешевле.

Поскольку опыт экспертов в сфере имитационного моделирования комплексных проектов увеличивается, в будущем предвидится меньше срывов проектов, перерасходов средств и при управлении проектами будет допускаться меньше ошибок.

СПИСОК ЛИТЕРАТУРЫ

• AnyLogic Company. 2018. Multimethod Simulation Modeling for Business Applications. Overview with Guided Model Building Example.

• Cooper, K. G. 1980. Naval Ship Production: A Claim Settled and a Framework Built. *Interfaces* 10, no. 6 (December): 20–36.

• Cooper, K. G. 1994. The $2,000 hour: how managers influence project performance through the rework cycle. *Project Management Journal* 25, no. 1: 11–24.

• Godlewski, E., G. Lee and K. Cooper. 2012. System Dynamics Transforms Fluor Project and Change Management. *INFORMS Journal on Applied Analytics* 42, no. 1: 17–32.

• Howick, S. 2003. Using System Dynamics to Analyse Disruption and Delay in Complex Projects for Litigation: Can the Modelling Purposes Be Met? *The Journal of the Operational Research Society* 54, no. 3: 222–229.

• Lyneis, J. M., K. G. Cooper and S. A. Els. 2001. Strategic management of complex projects: a case study using system dynamics. *System Dynamics Review* 17, no. 3 (Autumn): 237–260.

• Lyneis, J. M. and D. N. Ford. 2007. System dynamics applied to project management: a survey, assessment, and directions for future research. *System Dynamics Review* 23, no. 2–3 (Summer – Autumn): 157–189.

• Lyneis, J., O. de Weck and S. Eppinger. 2003. Course notes for MIT course ESD.36J: System and Project Management (Fall 2003).

• Reichelt, K. and J. Lyneis. 1999. The dynamics of project performance: benchmarking the drivers of cost and schedule overrun. *European Management Journal* 17, no. 2 (April): 135-150.

• Repenning, N. and J. Sterman. 2001. Nobody Ever Gets Credit for Fixing Problems that Never Happened: Creating and Sustaining Process Improvement. *California Management Review* 43, no. 4 (Summer): 64–88.

• Rodrigues, A. and J. Bowers. 1996. System dynamics in project management: A comparative analysis with traditional methods. *System Dynamics Review* 12, no. 2 (Summer): 121–139.

• Abdel-Hamid, T. and E. Madnick. 1982. Modeling the Dynamics of Software Project Management. Working paper, Alfred P. Sloan School of Management (February 1988, #WP 1980-88).

• Abdel-Hamid, T. K. 1993. A multiproject perspective of single-project dynamics. Journal of Systems Software 22, no. 3: 151–165.

• Andersen, D. F., J. A. M. Vennix, G. P. Richardson and E. A. J. A. Rouwette. 2007. Group model building: problem structuring, policy simulation and decision support. The Journal of the Operational Research Society 58, no. 5 (May): 691–694.

• Arnold, R.D. and J.P. Wade. 2015. A Definition of Systems Thinking: A Systems Approach. Procedia Computer Science 44: 669–678.

• Boehm, B. W. Software Engineering Economics. 1981. Upper Sadle River, New Jersey: Prentice-Hall PTR.

• Borshchev, A. 2013. The Big Book of Simulation Modeling: Multimethod Modeling with AnyLogic 6. La Vergne, TN: Lightning Source Inc.

• Chan, M. 2011. Fatigue: the most critical accident risk in oil and gas construction. Construction Management and Economics 29, no. 4: 341–353.

• Cooper, K. G. 1980. Naval Ship Production: A Claim Settled and a Framework Built. Interfaces 10, no. 6 (December): 20–36.

• Cooper, K. G. 1993a. The rework cycle: why projects are mismanaged. PM Network 7, no. 2: 5–7.

• Cooper, K. G. 1993b. The rework cycle: how it really works . . . and reworks . . . PM Network 7. no. 2: 25–28.

• Cooper, K. G. 1993c. The rework cycle: benchmarks for the project manager. Project Management Journal 24, no. 1: 17–21.

• Cooper, K. G. 1994. The $2,000 hour: how managers influence project performance through the rework cycle. Project Management Journal 25, no. 1: 11–24.

• Cooper, K. G. and K. S. Reichelt. 2004. Project changes: sources, impacts, mitigation, pricing, litigation, and excellence. In The Wiley Guide to Managing Projects, ed. P. W. G. Morris and J. K. Pinto, 743–772. Hoboken, NJ: Wiley.

• Cooper, K. G., J. M. Lyneis and B. J. Bryant. 2002. Learning to learn, from past to future. International journal of project management 20, no. 3 (April): 213–219.

• Dörner, D. 1996. The Logic of Failure: Recognizing and Avoiding Error in Complex Situations, trans. R. and R. Kimber. New York: Basic Books.

• Eden, C., T. Williams and F. Ackermann. 2005. Analysing project cost overruns: comparing the “measured mile” analysis and system dynamics modelling. International Journal of Project Management 23, no. 2 (February): 135–139.

• Eick S. G., C. R. Loader, M. D. Long, L. G. Votta and S. V. Wiel. 1992. Estimating software fault content before coding. In ICSE ‘92 Proceedings of the 14th international conference on Software engineering (Melbourne, Australia — May 11–15, 1992), 59–65. New York: ACM.

• Ford, D. N., J. M. Lyneis and T. R. B. Taylor. 2007. Project Controls to Minimize Cost and Schedule Overruns: A Model, Research Agenda, and Initial Results. In Proceedings of the 25th International Conference of the System Dynamics Society and 50th Anniversary Celebration, ed. J. Sterman, R. Oliva, R. S. Langer, J. I. Rowe and J. M. Yanni, 1–27. Albany, New York: System Dynamics Society.

• Ford, D.N. and J. Sterman. 1998. Dynamic modeling of product development processes. System Dynamics Review 14, no. 1 (Spring): 31–68.

• Ford, D.N., S. Anderson, A. Damron, R. de Las Casas, N. Gokmen, and S. Kuennen. 2004. Managing Constructability Reviews to Reduce Highway Project Durations. Journal of Construction Engineering and Management 130, no. 1 (February): 33–42.

• Godlewski, E., G. Lee and K. Cooper. 2012. System Dynamics Transforms Fluor Project and Change Management. INFORMS Journal on Applied Analytics 42, no. 1: 17–32.

• Graham, A. K. 2000. Beyond PM 101: Lessons for Managing Large Development Programs. Project Management Journal 31, no. 4: 7–18.(дата обращения: 12 марта 2019).

• Holzmann, G. 2009. Appendix D – Software Complexity. In NASA Office of Chief Engineer. NASA Study of Flight Software Complexity: Final Report.

• Holzmann, G. J. 2007. Conquering Complexity. Computer 40, no. 12 (December): 111–113.

• Hovmand, P.S. 2014. Community Based System Dynamics. New York: Springer Science+Business Media.

• Lane, D. C. 2000. Diagramming conventions in system dynamics. The Journal of the Operational Research Society 51, no. 2 (February): 241–245.

• Larson, W.J. and J.R. Wertz. 1993. Space Mission Analysis and Design. 2nd ed. Torrance, California: Microcosm.

• Lee Z. W., D. N. Ford and N. Joglekar. 2007. Resource allocation policy design for reduced project duration: a systems modeling

approach. Systems Research and Behavioral Science 24, no. 6 (November/December): 551–566.

• Love, P.E.D., G.D. Holt, L.Y. Shen, H. Li, Z. Irani. 2002. Using systems dynamics to better understand change and rework in construction project management systems. International Journal of Project Management 20, no. 6 (August): 425–436.

• Lyneis, J. M. and D. N. Ford. 2007. System dynamics applied to project management: a survey, assessment, and directions for future research. System Dynamics Review 23, no. 2–3 (Summer – Autumn): 157–189.

• Lyneis, J. M., K. G. Cooper and S. A. Els. 2001. Strategic management of complex projects: a case study using system dynamics. System Dynamics Review 17, no. 3 (Autumn): 237–260.

• NASA Office of Chief Engineer. 2009. NASA Study of Flight Software Complexity: Final Report.

• Neves, F. G., H. Borgman and H. Heier. 2016. Success Lies in the Eye of the Beholder: The Mismatch Between Perceived and Real IT Project Management Performance. In 49th Hawaii International Conference on System Sciences, 5878–5887. Los Alamitos, California: IEEE Computer Society.

• Pena-Mora, F. and M. Park. 2001. Dynamic Planning for Fast-Tracking Building Construction Projects. Journal of Construction Engineering and Management 127, no. 6 (December): 445–456.

• Richardson, G. P. and A. Pugh III. 1981. Introduction to System Dynamics Modeling with Dynamo. Cambridge, MA: MIT Press.

• Richardson, G. P. and D. F. Andersen. 1995. Teamwork in group model building. System Dynamics Review 11, no. 2 (Summer): 113–137.

• Richardson, G.P. and D.F. Andersen. 2010. Systems Thinking, Mapping, and Modeling in Group Decision and Negotiation. In Handbook of Group Decision and Negotiation, ed. D. M. Kilgour and C. Eden, 313–324. Dordrecht: Springer Verlag

• Roberts, E. B. 1964. The Dynamics of Research and Development. New York: Harper & Row.

• Roberts, E. B. 1974. A simple model of R&D project dynamics. R&D Management 5, no. 1: 1–15.

• Rodrigues, A. and J. Bowers. 1996. The role of system dynamics in project management. International Journal of Project Management 14, no. 4 (August): 213–220.

• Schelling, T.C. 1978. Micromotives and Macrobehavior. New York: W. W. Norton & Co.

• Simon, H. A. 1972. Theories of Bounded Rationality. In Decision and Organization: A Volume in Honor of Jacob Marschak, ed. C. B. McGuire and R. Radner, 161-176. Amsterdam: North-Holland.

• Simon, H. A. 1979. Rational Decision Making in Business Organizations. The American Economic Review 69, no. 4 (September): 493–513.

• Stave, K. 2010. Participatory System Dynamics Modeling for Sustainable Environmental Management: Observations from Four Cases. Sustainability 2, no. 9: 2762–2784.

• Stecklein, J. M., J. Dabney, B. Dick, B. Haskins, R. Lovell and G. Moroney. 2004. Error Cost Escalation Through the Project Life Cycle.

• Sterman, J. 1992. System Dynamics Modeling for Project Management.

• Taylor, T. and D. N. Ford. 2006. Tipping point failure and robustness in single development projects. System Dynamics Review 22, no. 1 (Spring): 51–71.

• The AnyLogic Company. 2016. Developing Disruptive Business Strategies with Simulation.

• Van den Belt, M. 2004. Mediated Modeling: A System Dynamics Approach to Environmental Consensus Building. Washington, DC: Island Press.

• Vennix, J. A. M. 1996. Group Model Building: Facilitating Team Learning Using System Dynamics. Chichester: Wiley.

• Williams, T. M. 1999. The need for new paradigms for complex projects. International Journal of Project Management 17, no. 5 (October): 269–273.

• Williams, T., C. Eden, F. Ackermann and A. Tait. 1995. The Effects of Design Changes and Delays on Project Costs. Journal of the Operational Research Society 46, no. 7 (July): 809–818.

ДОПОЛНИТЕЛЬНЫЕ РЕСУРСЫ

- Истории успеха
- Избранные статьи
- Обучающие видео
- Книги
- Скачать AnyLogic
- Тренинги и события

СВЯЗАТЬСЯ С НАМИ

Компания AnyLogic – офис в России

info@anylogic.com