Масштабированный скрам

Как организовать
гибкую разработку
в крупной компании

Крэг Ларман
Бас Водде

LeSS С 2005 года мы помогаем нашим клиентам применять масштабируемый скрам (LeSS) для бережливой и гибкой разработки в больших продуктовых группах. Мы делимся этим опытом и знаниями, чтобы вы тоже могли добиться успеха.

альпина паблишер | бизнес

Гибкие методы управления

Бас Водде

Масштабированный скрам. Как организовать гибкую разработку в крупной компании

«Альпина Диджитал»

2009

Водде Б.

Масштабированный скрам. Как организовать гибкую разработку в крупной компании / Б. Водде — «Альпина Диджитал», 2009 — (Гибкие методы управления)

ISBN 978-5-96-148396-3

Эксперты по масштабированному скраму (LeSS) Крэг Ларман и Бас Водде предлагают читателям своей книги полный инструментарий для использования скрама в больших и распределенных группах разработки. Опираясь на свой и чужой опыт внедрения гибкой разработки, авторы собрали под одной обложкой ответы на ключевые вопросы, которые могут возникнуть у агента изменений, решившего трансформировать свою команду и сделать ее непобедимой. Вы узнаете, как обеспечивать устойчивость кода, когда проводить тестирование, по каким метрикам оценивать эффективность процессов, устанавливать систему вознаграждений и определять требования к продукту на разных этапах разработки. Отдельный раздел посвящен разбору скрам-терминологии: прочитав его, вы усвоите, что такое спринт, чем занимается скрам-мастер и кто отвечает за бэклог продукта.

ISBN 978-5-96-148396-3

© Водде Б., 2009
© Альпина Диджитал, 2009

Содержание

Крэг Ларман, Бас Водде Масштабированный скрам: Как организовать гибкую разработку в крупной компании

Переводчик *Ирина Евстигнеева*

Редактор *Алексей Воронин*, эксперт в трансформациях крупных организаций, сертифицированный тренер по LeSS

Редактор *Любовь Любавина*

Главный редактор *С. Турко*

Руководитель проекта *А. Василенко*

Дизайн обложки *Ю. Буга*

Художественное оформление и макет *Ю. Буга*

Корректоры *Е. Аксёнова, Т. Редькина*

Компьютерная верстка *М. Поташкин*

Все права защищены. Данная электронная книга предназначена исключительно для частного использования в личных (некоммерческих) целях. Электронная книга, ее части, фрагменты и элементы, включая текст, изображения и иное, не подлежат копированию и любому другому использованию без разрешения правообладателя. В частности, запрещено такое использование, в результате которого электронная книга, ее часть, фрагмент или элемент станут доступными ограниченному или неопределенному кругу лиц, в том числе посредством сети интернет, независимо от того, будет предоставляться доступ за плату или безвозмездно.

Копирование, воспроизведение и иное использование электронной книги, ее частей, фрагментов и элементов, выходящее за пределы частного использования в личных (некоммерческих) целях, без согласия правообладателя является незаконным и влечет уголовную, административную и гражданскую ответственность.

© 2009 by Pearson Education, Inc.

Authorized translation from the English language edition, entitled Lean & Agile Development: Thinking and Organizational Tools for Large-Scale Scrum, 1st edition by Craig Larman; Bas Vodde, published by Pearson Education, Inc., publishing as Addison-Wesley professional.

© Издание на русском языке, перевод, оформление. ООО «Альпина Паблишер», 2023

* * *

Крэг Ларман
Бас Водде

Масштабированный скрам

Как организовать гибкую разработку в крупной компании

Перевод с английского

альпина
паблишер
Москва
2023

Нашим клиентам,

моему другу и соавтору Басу

и 孙媛

Предисловие

Мы благодарим вас за то, что вы решили прочитать эту книгу! Мы постарались сделать ее полезной. Дополнительные материалы и рекомендации можно найти на сайтах www.craiglarman.com. и www.odd-e.com. Пожалуйста, свяжитесь с нами, если у вас возникнут вопросы.

Условные обозначения в тексте

Момент, который нужно подчеркнуть, или *не очень существенный новый термин*. **Важная идея** или **важный новый термин**. [Bob67] – ссылка на библиографию.

Об авторах

Крэг Ларман – научный директор в международной консалтинговой и аутсорсинговой компании Valtech, имеющей подразделения в Европе, Азии и Северной Америке. Как консультант и коуч в области менеджмента и разработки работает с крупными и офшорными продуктовыми группами, помогая им с внедрением гибких и бережливых методологий разработки преимущественно с акцентом на встраиваемых системах. Руководил переходом на гибкую разработку (на основе скрама) в Valtech India; был автором и ведущим коучем в проекте по внедрению бережливой разработки ПО в компании Xerox. На протяжении многих лет в качестве консультанта и коуча помогал с крупномасштабным внедрением гибкой разработки и скрама в компаниях Nokia, Siemens, NSN и многих других. Родился в Канаде, с 1978 г. периодически живет в Индии. Автор книг «Гибкая и итеративная разработка: руководство для менеджера» (Agile and Iterative Development: A Manager's Guide and Applying UML) и «Применение UML 2.0 и шаблонов проектирования»[1] – самых продаваемых в мире книг по гибкой и итеративной разработке и объектно-ориентированному проектированию и дизайну (OOA/D). Также вместе с Басом Водде написал книгу «Практики масштабирования бережливой и гибкой разработки: Масштабированный скрам для больших, распределенных и офшорных продуктовых групп» (Practices for Scaling Lean & Agile Development: Large, Multisite, and Offshore Product Development with Large-Scale Scrum).

В юности мечтал стать странствующим уличным музыкантом, но ему не хватило таланта, поэтому он получил степень бакалавра, а затем магистра компьютерных наук в ведущем канадском Университете имени Саймона Фрейзера в Ванкувере. В 1970-х гг. занимался разработкой систем на языках APL и 4GL, а в начале 1980-х заинтересовался искусственным интеллектом (вследствие недостатка собственного).

Бас Водде – независимый консультант и коуч по разработке продуктов и использованию масштабированного скрама. В течение нескольких лет руководил в Nokia Networks внедрением гибкой разработки и скрама в масштабах всей компании. Входил в руководящую команду очень крупной распределенной командой разработки (работающей в Европе и Китае), осуществлявшей переход на скрам. Работал старшим разработчиком/архитектором в области встраиваемых телекоммуникационных систем, а также менеджером по качеству. Руководил разработкой решений и коучингом в проектах разработки через тестирование (TDD) встраиваемых систем. Вместе с Крэгом Ларманом написал книгу «Практики масштабирования бережливой и гибкой разработки». Родился в Голландии, много лет жил в Китае, сейчас живет в Сингапуре.

[1] Ларман К. Применение UML 2.0 и шаблонов проектирования: Введение в объектно-ориентированный анализ, проектирование и итеративную разработку. – Киев: Диалектика, 2019.

Благодарности

Прежде всего мы хотим сказать спасибо всем нашим клиентам.

Мы также хотим поблагодарить наших рецензентов и помощников: Питера Альфвина, Алана Атласа, Габриэль Бенефилд, Бьярте Богснес, Майка Бриа, Ларри Кая, Майка Кона, Пита Демера, Эстер Дерби, Ютту Экштейн, Кенджи Хиранабе, Клинтона Кита, Куроива-сан, Диану Ларсен, Тимо Леппянена, Эрика Линдли, Мэри Поппендик, Тома Поппендика, Кена Швабера, Маартена Смитса, Джеффа Сазерленда, Дэйва Томаса и Вилле Валтонена.

Нынешних и бывших членов команды (и рецензентов) компании Flexible, в том числе Кати Вилки, Петри Хаапио, Лассе Коскела, Пола Наги, Йоонаса Рейндерса, Габора Гунихо, Сами Лилья и Ари Тикка. И нынешних и бывших членов команды (и рецензентов) компании IPA LT, особенно Теро Пелтола и Люй И.

Бас признателен своей жене Сунь Юань за ее долготерпение в те периоды жизни, когда он «сосредоточивается на книгах», а не на ней, и за ее поддержку в ходе многочисленных переездов и разъездов по разным странам, где находятся его корпоративные клиенты. А также он благодарен Крэгу за стимулирующие дискуссии и годы совместной работы над крупными проектами, включая «отладку» организаций, и над книгами.

Крэг благодарит Альбертину за поддержку его писательской деятельности.

Мы также благодарим Луизу Адэр, Райну Хробак, Криса Гузиковски, Джули Нахил и Мэри Лу Нор за помощь в издании этой книги.

Глава 1
Введение

Будущее теперь уже не то, что раньше.
Йоги Берра

Мы сидели в переговорной комнате и пили горячий кофе. За окном – типичное для Северной Европы пронизывающее холодом зимнее утро. Вскоре вошел наш новый клиент и пожал нам руки. «Спасибо, что приехали, – сказал он. – Начнем с того, что у нас довольно небольшая группа разработки, человек 80, плюс-минус».

Однажды мы помогали с внедрением гибкой разработки в команде, которая мечтала вырасти до *очень* большого размера: до 12 человек.

У людей разные шкалы того, что значит «большая» продуктовая группа[2]. Для кого-то это 50 человек или даже меньше. Для других – в разы больше. Мы обычно работаем с однопродуктовыми группами, насчитывающими 100–500 человек (которые внедряют принципы скрама, бережливого мышления и гибкой разработки, как правило, в области программно-интенсивных встраиваемых систем). Вот наш подход: *«Если собрать всю группу в одной комнате, вы сможете вспомнить имена всех ее членов? Если нет, значит, у вас большая группа».* Таково наше определение «большой» продуктовой группы, основанное на фундаментальном критерии: нашей ограниченной памяти.

Наша ключевая рекомендация: проработав много лет с большими, распределенными и офшорными (удаленными) продуктовыми группами, мы проанализировали наш опыт и сделали следующий вывод: *не работайте с такими группами*!

Есть гораздо более эффективные способы разработки больших продуктов, чем задействовать рой разработчиков, рассеянных по разным местам. Лучше создать небольшую группу из крутых разработчиков и других специалистов, способных работать командами, хорошо им платить и разместить их в одном месте с продуктовым менеджером, который выступает в качестве «голоса клиента».

Но мы знаем, что вы не прислушаетесь к нашему совету и *все равно* будете использовать большие, распределенные и офшорные группы разработки. Причина в том, что ваша существующая система уже структурирована таким образом, или в том, что вы придерживаетесь (пред)убеждения, будто «большие системы требуют большого количества людей». Наши клиенты регулярно спрашивают нас: «Как рассчитать, сколько людей нам может потребоваться?» Мы всегда советуем: «Начните с небольшой группы сильных специалистов и увеличивайте ее только тогда, когда без этого действительно невозможно обойтись». Такая необходимость возникает редко.

Раз уж вы *все равно* будете использовать большие, распределенные и офшорные группы, мы хотим поделиться с вами нашим собственным и чужим опытом по применению принципов и практик бережливой и гибкой разработки в такой рабочей среде[3].

---

[2] Методология скрама (и эта книга) применима к разработке продуктов как для внешнего рынка, так и для внутреннего пользования.

[3] В книге «Практики масштабирования бережливой и гибкой разработки: Масштабированный скрам для больших, рассредоточенных и офшорных продуктовых групп» содержатся детальные практические советы касательно масштабирования и планирования, продукт-менеджмента, распределенной и офшорной разработки, контрактов, требований, проектирования и архитектуры, координации, унаследованных кодов, тестирования и многого другого.

Инструменты мышления и организации

Когда Бас входил в руководящую команду одной большой продуктовой группы, он постоянно во время совещаний спрашивал: «Почему мы делаем это именно так? Что будет, если мы сделаем это иначе?» Через несколько месяцев один из членов команды признался Крэгу: «Первое время эти вопросы выводили меня из себя. Но потом я оценил их значение». Бас не стремился вывести людей из себя; он хотел побудить их к *системному мышлению*, позволяющему: 1) увидеть более глубокую динамику организационной системы в целом; 2) понять, как система стала такой, какая она есть; и 3) пересмотреть предположения, лежащие в основе существующей организации.

Когда люди впервые слышат о скраме с его короткими, ограниченными по времени итерационными циклами разработки, они поначалу воспринимают это как локализованную практику инкрементального создания продукта за счет небольших управляемых шагов с использованием обучения и корректировки, которые ведут к поставленной цели. Поэтому они говорят: «О, аджайл меня не касается. Это для *разработчиков*». Но запуск таких циклов обучения на нижнем организационном уровне потенциально способен оказать гораздо более масштабное воздействие: как правило, это требует запуска аналогичных «петель» обучения и на более высоком уровне – создания обучающейся организации, где люди будут постоянно пересматривать структуры и правила, которые определяют и окружают разработку продуктов. Внедрение принципов скрама или бережливого подхода в очень больших продуктовых группах неизбежно ведет к такой высокоуровневой организационной трансформации.

Пример. Представим компанию, в которой подразделение разработки внедряет скрам, чтобы повысить адаптивность. Но отдел продаж продолжает работать по старинке: его сотрудники стремятся увеличить личные комиссионные и ежеквартальные продажи, с почти безграничным оптимизмом расписывая клиентам то, что «могут сделать наши самые лучшие разработчики», и обещая им достать звезды с неба. В результате подразделение разработки сталкивается с «обязательствами», которые оно не давало и не в состоянии выполнить; в компании его обвиняют в неисполнении «*наших* обещаний» и делают вывод, что «скрам не работает».

Если бы эта книга была посвящена внедрению скрама в небольшой однопродуктовой команде из двух десятков человек, системное мышление и организационные инструменты были бы интересными, но не критически важными темами. Однако они играют ключевую роль для успешного внедрения скрама, если данная методология масштабируется на однопродуктовую группу, скажем, из 400 человек, которая, возможно, работает в составе крупной компании-разработчика, насчитывающей тысячи сотрудников, и совершает переход к скраму в тесном сотрудничестве с отделами продаж и доставки в условиях ограничений, налагаемых традиционными корпоративными правилами в отношении управления человеческими ресурсами, структуры команд, отчетности, оценки эффективности, управления проектами, контрактов и вознаграждения.

Следовательно, эта книга утверждает, что один из краеугольных камней успешного масштабирования скрама и гибкой разработки – люди, которые готовы учиться и применять необходимые *инструменты мышления*[4], включая системное мышление, анализ ментальных моделей, бережливое мышление, теорию массового обслуживания и распознавание ложных дихотомий (но не ограничиваясь этим).

Применение этих инструментов мышления позволяет увидеть, когда и где существующая организационная структура препятствует потоку создания ценности и требует реорганизации.

[4] Термин «инструменты мышления» был популяризирован в [Reinertsen97].

Следовательно, второй краеугольный камень успешного масштабирования скрама, который рассматривается в этой книге, – *инструменты организации*, включая фича-команды, области требований, а также многие другие изменения в структуре, процессах, задачах, управлении людьми и системе вознаграждения.

Инструменты действия

Наряду с инструментами мышления и организации для успешной работы большой, распределенной и офшорной продуктовой группы требуются *инструменты действия* – конкретные практики разработки. Эффективное использование этих инструментов (которые подробно описаны в нашей книге «Практики масштабирования бережливой и гибкой разработки») отчасти зависит от организационной трансформации. Многие практики могут применяться и без более глубоких структурных изменений, но их эффективность в этом случае будет ограничена.

Таким образом, внедрение инструментария, описанного в этой книге, можно рассматривать как предварительное условие применения инструментов действия (практик), представленных в книге «Практики масштабирования бережливой и гибкой разработки». Но на деле *практики* всегда внедряются первыми – потому что людям проще начинать именно с них. Тем не менее в какой-то момент вам все равно придется вернуться назад и воспользоваться организационными и мыслительными инструментами.

Мы рекомендуем коучам и другим агентам изменений, участвующим во внедрении скрама или бережливой разработки на больших масштабах, начать с внедрения в команде инструментов действия и параллельно самим как следует изучить инструменты мышления и организации. В какой-то момент ситуация созреет и люди будут готовы перейти от обсуждения «Как мы будем осуществлять полномасштабную непрерывную интеграцию?» к вопросам: *«Не мешают ли существующие правила управления персоналом созданию эффективных команд?»* и *«Каков в данном случае поток создания ценности и как ему может препятствовать существующая организационная структура?»*.

Эксперименты: «Попробуйте… Избегайте…»

Скрам делает упор на эмпирический контроль процесса; в процессах разработки слишком высок уровень сложности и вариабельности, чтобы использовать шаблонные подходы. Поэтому инструменты в обеих наших книгах представлены в виде советов, начинающихся со слов: «Попробуйте…» или «Избегайте…». Это всего лишь эксперименты, и некоторые из них, вероятно, не сработают в ваших конкретных обстоятельствах. Подход в скраме, как и в бережливом мышлении *кайдзен,* состоит в том, чтобы сначала изучить и понять существующую ситуацию (*инспектировать*), а затем с помощью экспериментов найти способы, как ее улучшить (*адаптировать*). Нацеленность на непрерывное экспериментирование – один из столпов бережливого подхода. На самом деле самый неудачный эксперимент – тот, который не был проведен. В Toyota Тайити Оно, ключевой создатель концепции бережливого подхода, шел непосредственно на место и инспектировал все документы с описанием стандартов. Если те были в буквальном или переносном смысле покрыты пылью, давно не менялись, он заставлял людей пересмотреть их. Непрерывная эволюция «стандартов» была его требованием.

В скраме этот цикл инспекции и адаптации (на основе экспериментов) повторяется в каждой двух- или четырехнедельной итерации на протяжении всего времени, пока существует продукт. А в бережливом подходе этот цикл постоянного экспериментирования и улучшения применяется как к отдельным продуктам, так и ко *всей организации в целом.*

Ограничения

Для нас есть еще большое поле для исследования и изучения в этой теме. В основе этой книги лежат наши сегодняшние (довольно ограниченные) опыт и понимание, которые мы рассчитываем углубить в ближайшие годы. Например, прожив несколько лет в Китае и Индии, мы только-только затронули верхушку в такой важной области, как мультикультурность в офшорных и распределенных группах разработки. Тем не менее мы надеемся на то, что наши советы окажутся для вас полезными, и будем благодарны, если вы поделитесь с нами своими историями и идеями.

Масштабное внедрение скрама способно повлиять практически на все аспекты организации и деятельности продуктоориентированной компании. Но, чтобы эта книга не получилась слишком пространной, а также из-за нашего ограниченного опыта в некоторых областях, мы лишь поверхностно затронули или же вовсе не стали рассматривать некоторые темы, которые, безусловно, заслуживают более пристального внимания, а именно:

- бюджетирование и финансы;
- продажи;
- маркетинг;
- разработка аппаратного обеспечения;
- разработка не-ИТ-продуктов;
- развертывание/поставка;
- сервисная поддержка.

В сущности, все написанное в этой книге применимо к разработке любых продуктов в целом. Скрам и бережливая разработка не ограничиваются только программным обеспечением [NT86]. Акцент на преимущественно программных системах (как правило, встраиваемых) объясняется только нашей сферой специализации, а также неуклонно растущей распространенностью ПО в повседневной жизни, включая стиральные машины и даже обувь.

В частности, в этой книге мы рассматриваем некоторые принятые в традиционных организациях предположения и правила, которые препятствуют *потоку создания ценности* и *эффективности команд*. В некоторых компаниях такой анализ может стать очень серьезным вызовом. Мы не хотим вас пугать, но организационная перестройка, призванная обеспечить поддержку бережливой и гибкой разработки, невозможна без отказа от многих традиционных моделей и радикального увеличения прозрачности. Организационные изменения также могут привести к ненужности старых ролей и перемещению с них талантливых людей. Как и в Toyota, мы поощряем поиск новых сфер приложения сил для компетентных людей, во-первых, потому что они этого заслуживают, а во-вторых, потому что в противном случае это будет тормозить перемены.

Это большая книга. Мы сожалеем, что, несмотря на все наши старания, не смогли сделать обсуждение масштабированного скрама… чуть менее масштабным.

Итак, поговорим об инструментах мышления.

Инструменты мышления
Глава 2
Системное мышление

Я прошел курс скорочтения и прочитал роман «Война и мир» за 20 минут. Он про Россию.
Вуди Аллен

«Что бы мы ни делали, количество дефектов в нашем бэклоге остается примерно тем же», – пожаловался нам один менеджер. Речь шла о продукте, насчитывающем 15 млн строк исходного кода C и C++, над которым работала группа из нескольких сотен разработчиков, внедрявшая принципы бережливой разработки. В чем же дело? Ответить на этот вопрос может только системное мышление. В небольших группах все действующие силы легко увидеть и понять без каких-либо формальностей, но при разработке крупного продукта – и в любых больших системах вообще – может быть действительно трудно. Джерри Вайнберг в этой ситуации выделяет два решающих фактора:

> *Закон Вайнберга – Брукса: от действий менеджеров, основанных на* ***неправильных моделях системы,*** *пострадало больше проектов разработки программных продуктов, чем от всех остальных причин, вместе взятых.*
>
> *Ловушка причинно-следственной связи: у каждого следствия есть причина… и мы всегда можем точно определить причину и ее следствие [Weinberg92].*

Эти факторы отражают влияние на систему наших **ментальных моделей**, о чем мы поговорим в этой главе чуть позже.

Проблемы, проистекающие из ментальных моделей и предположений, – это только один аспект. Другой состоит в том, что крупномасштабное внедрение скрама, принципов бережливого подхода и гибкой разработки не происходит изолированно только в группе разработки. Это затрагивает и такие области, как управление продуктами, бюджетирование, бета-тестирование, запуск, корпоративное управление и управление персоналом. Следовательно, при крупномасштабном внедрении гибкого фреймворка не просто полезно, но и необходимо привлечь людей из этих областей к тому, чтобы *эффективно поразмышлять* о ментальных моделях, причинно-следственных связях, петлях обратной связи и механизмах контроля (или иллюзии контроля) в большой системе, которая вскоре будет серьезно *нарушена*. Системное мышление – инструментарий, который поможет это сделать.

Стоит ли полагаться на здравый смысл?

«Здравый смысл говорит нам…» – эту фразу часто можно услышать не только в повседневной жизни, но и на встречах скрам-команд. Но что такое здравый смысл? Эйнштейн дал ему такое определение: *«Здравый смысл – это сумма предубеждений, приобретенных человеком к 18 годам»*.

Тайити Оно, отец производственной системы Toyota, писал: «*Ошибочные представления легко превращаются в здравый смысл. Тогда споры становятся бесконечными. Каждая сторона пытается переспорить другую, и дело не движется с места. Вот почему [у нас в компании] был период, когда я постоянно говорил людям «выйти за рамки здравого смысла» и начать думать вне их. В рамках здравого смысла некоторые вещи могут казаться нам правильными, но только лишь из-за встроенных*

в него заблуждений. Кроме того, зачастую мы считаем необходимым руководствоваться здравым смыслом только по одной причине: потому что многолетний опыт показал, что данный образ, хотя и не несет больших преимуществ, не имеет и серьезных недостатков… Все мы – люди, а людей можно сравнить с ходячими наборами заблуждений, которые верят, что привычный способ делать вещи – лучший. И даже если вы не считаете его действительно лучшим, устоявшийся здравый смысл говорит вам: «Мы ничего не можем с этим поделать, так обстоят дела» [Ohno07].

«Здравый смысл» не самый надежный способ понять нелинейные системы, такие как крупномасштабная разработка продуктов.

В 1958 г. в журнале *Harvard Business Review* была опубликована новаторская статья «Индустриальная динамика: важный прорыв для принимающих решения лиц» Джея Форрестера, профессора Школы менеджмента Слоуна Массачусетского технологического института (МТИ) [Forrester58]. Эта статья положила начало преподаванию **системной динамики** в Школе менеджмента Слоуна, а вскоре обучение системному мышлению стало неотъемлемой частью бизнес-образования. Системная динамика иногда трактуется как синоним **системного мышления**, но последнее более общий термин.

В МТИ работали и другие выдающиеся исследователи в области системной динамики, например Питер Сенге[5].

В соответствии с законом Вайнберга–Брукса, исследование Форрестера показало, что, когда принимающим решения лицам представляли динамические модели бизнес-систем и предлагали улучшить их производительность, они *обычно делали их только хуже* [SKRRS94]. Исследование также показало, что большинство людей имеют слабое представление о том, как можно фундаментально улучшить системную динамику, и обычно применяют ошибочный «здравый смысл» и «быстрые решения», которые не приводят к устойчивым системным улучшениям.

Почему поведение большой группы разработки (как некой системы) так трудно понять и еще труднее грамотно им руководить? Ответ отчасти кроется в поведении стохастических систем с очередями и изменчивостью, которое будет рассмотрено в главе о теории массового обслуживания. Отчасти – в *теории управления*: большинство интересующих нас систем (например, большая группа разработки продукта) имеют сложные петли положительной и отрицательной обратной связи и нелинейное поведение. Поведение этих систем зачастую не поддается интуитивному пониманию. И наконец, отчасти – в такой «второстепенной» проблеме, как *люди*.

Таким образом, основные причины неспособности понять большую систему и грамотно ею руководить включают следующие факторы (но не ограничиваются ими):

- недостаток знаний о системной динамике, петлях обратной связи, поведении нелинейных систем и непредвиденных воздействиях в системах рабочей среды;
- непонимание корневых причин проблем (и того, как их найти) – именно *причин*, а не одной причины: системное мышление позволяет увидеть, что у проблем множественные, косвенные и динамические причины;
- непонимание того, как и почему «быстрые решения» или локальные решения на уровне отделов могут вести к ухудшению общей производительности системы.

[5] Сенге написал книгу «Пятая дисциплина», посвященную системному мышлению и обучающимся организациям, которую журнал *Harvard Business Review* назвал одной из основополагающих книг по менеджменту за последние 75 лет. См. [Senge94].

Короче говоря, все дело в неспособности мыслить системно[6].

Эти причины могут иметь серьезные последствия, когда широкомасштабное внедрение принципов бережливого и гибкого подходов пересекаются с традиционным менеджментом. Управленческая команда является частью системы, которая подвергается трансформативному нарушению, и, если ее члены не применяют системное мышление, они сами могут стать серьезным нарушающим фактором – в плохом смысле.

В качестве краткого описания системного мышления нам нравятся следующие 11 «законов», описанных в книге Питера Сенге «Пятая дисциплина»:

1. Сегодняшние проблемы вызваны вчерашними «решениями».
2. Сила воздействия на систему равняется силе ее противодействия.
3. Прежде чем улучшиться, поведение системы ухудшается.
4. Легкий путь обычно ведет назад.
5. Лечение может быть хуже болезни.
6. Тише едешь, дальше будешь.
7. Причины и следствия могут быть разделены во времени и пространстве.
8. Малые изменения могут привести к значительным результатам, но такие области с наивысшей отдачей не всегда очевидны.
9. Можно погнаться за двумя зайцами и поймать обоих – пусть не одновременно.
10. Разделив слона пополам, вы не получите двух маленьких слоников.
11. Не нужно искать виноватых.

Внутренний девиз Toyota: «Правильное мышление, хорошие продукты». Системное мышление – это *набор* инструментов мышления, которые помогают:

■ **Видеть системную динамику:**

○ организация, занимающаяся разработкой, – это система, образованная людьми и правилами, с неявными петлями обратной связи и непреднамеренными воздействиями;

○ мы можем научиться видеть эту динамику и, следовательно, улучшать систему с помощью д**иаграмм причинно-следственных циклов** (*causal loop diagrams*), создаваемых на групповых встречах.

■ **Видеть ментальные модели:**

○ одна из причин неоптимальных решений – ложные предположения и ошибочные выводы;

○ ментальные модели обнаруживаются с помощью диаграмм причинно-следственных циклов и метода «Пяти почему».

■ **Выявлять корневые причины:**

○ чтобы реально улучшить систему, необходимо научиться выявлять корневые причины проблем и видеть более глубокие связи;

○ выявить корневые причины помогают диаграммы причинно-следственных циклов, метод «Пять почему» и диаграммы Исикавы.

■ **Видеть локальную оптимизацию:**

○ еще один источник неоптимальных решений – **локальная оптимизация**, когда принимается решение, «лучшее» с точки зрения отдельного человека или отдела, в противовес **глобальной оптимизации,** исходящей из цели бережливого подхода системного уровня – *создавать ценность быстро, с высоким качеством и высоким моральным духом*.

---

[6] Еще одна причина – вера в возможность большего контроля, чем это возможно на самом деле. Наука о комплексных системах показывает существование фундаментальных ограничений в прогнозировании и управлении полухаотическими социальными системами [Stacey07]. Это довольно большой ящик Пандоры, который мы не будем открывать в этой книге.

Итак, эта глава построена вокруг следующих областей системного мышления: учимся видеть: (1) *системную динамику*, (2) *ментальные модели*, (3) *корневые причины* и (4) *локальную оптимизацию*.

Учимся видеть системную динамику

Статическая сложность против динамической сложности

Многие из нас, особенно инженеры и финансисты, обучены хорошо справляться со **статической сложностью** – решением таких задач, как анализ и управление информацией (требованиями, финансовыми данными и пр.), разбивка сложных структур на более простые и т. д., то есть со сложностью статического, информационного или структурного характера.

Почему большие программные системы имеют тенденцию деградировать, несмотря на то что на работу с дефектами тратится все больше и больше времени? Что может случиться, если США вторгнутся в Ирак? Чтобы увидеть динамику, стоящую за этими вопросами, требуется анализ **комплексной динамики**.

В отличие от обучения статическому анализу, многие из нас не получают *формального* образования в области анализа *динамической сложности*[7], особенно динамики рабочей среды. Возможно, причина этого – в распространенном мнении, что здесь достаточно полагаться на здравый смысл. Но Форрестер продемонстрировал, что «здравый смысл» не справляется с комплексными системами, и показал, что обучение людей формальным методам, в частности таким, как использование *моделей системной динамики*, визуализированных в виде *диаграмм потоков* [Forrester61], позволяет улучшить их способность к системному мышлению в рабочей среде.

Диаграмма потоков охватывает материальные, финансовые и информационные потоки, метрики (переменные, имеющие количественные значения, такие как финансовые средства или количество дефектов), последствия решений и правил, а также причинно-следственные связи. Популярное упрощение – **диаграмма причинно-следственных циклов**, которая фокусируется на действующих в системе причинно-следственных отношениях и петлях обратной связи [Sterman00]. Существует несколько вариантов нотаций таких диаграмм; во всех них присутствуют метрики (количественные переменные), причинно-следственные связи и задержки (отложенные эффекты). Вайнберг [Weinberg92] называет это *диаграммой влияний*.

Первый закон построения диаграмм: моделируйте, чтобы обсуждать

Чтобы научиться видеть системную динамику, используйте диаграммы причинно-следственных циклов, которые в идеале следует нарисовать на доске совместно с коллегами в ходе ретроспективы спринта. Но прежде чем двигаться дальше, давайте рассмотрим *первый закон построения диаграмм*:

> *Основная ценность диаграмм заключается в дискуссии, возникающей в процессе их построения: мы моделируем, чтобы инициировать обсуждение.*

Когда группа собирается, чтобы нарисовать на доске диаграмму причинно-следственных связей (рис. 2.1), основная ценность состоит в обсуждениях и общем понимании, которое зарождается в ходе построения модели. Визуализация в виде наглядной диаграммы также *важна*, чтобы сделать идеи конкретными и однозначными – явно отобразить ментальные модели, которые есть у участников, – потому что слова сами по себе могут быть расплывчатыми и неправильно интерпретируемыми. Тем не менее сама диаграмма вторична по отноше-

[7] Среди исключений – макроэкономика, психология, социология и биология.

нию к тому, что люди уносят с собой: новые идеи и более глубокое понимание, пересмотренное в результате обсуждения.

Рис 2.1. В Valtech India: дискуссии и осмысление — самое важное в процессе построения диаграмм

Базовые проблемы и простые удобные инструменты

С годами мы овладели очень продвинутыми, сложнейшими приемами анализа и разработки решений в области инженерии, менеджмента и т. д. Поначалу мы были вдохновлены этими инструментами и стремились применять их и обучать им других людей при любом случае, пока в ходе реальной работы не поняли, что:

> *подавляющее большинство проблем в бизнесе (в том числе в разработке) носят настолько базовый характер*[8]*, что ключевым решением является обучение и планомерное использование* ***простых и удобных*** *инструментов мышления и действия.*

Простота. Книги и курсы по моделированию системной динамики и построению диаграмм причинно-следственных циклов имеют тенденцию чрезмерно все усложнять, предлагая забираться в такие дебри, как концепция *архетипов*, описанная в «Пятой дисциплине», компьютерное моделирование, нелинейные уравнения и т. д. На практике это только отпугивает людей от использования этого, в сущности, очень простого инструмента: вам нужно всего лишь встать у доски, обсудить со своими коллегами, какие причинно-следственные связи и петли обратной связи действуют в вашей системе, и отобразить их в виде диаграммы на доске.

Когда вы выбираете инструменты мышления или действия для использования в рабочей среде, помните слова Вольтера:

---

[8] «Базовая» не означает простая или неважная. Например, мотивация и качество – это базовые, но далеко не простые и не пустяковые проблемы.

«Le mieux est l'ennemi du bien» («Лучшее – враг хорошего»).

Удобство. Существует множество изощренных инструментов мышления и действия, которые так любят теоретики, но которые почему-то никак не приживаются на практике. Вместе с тем практики скрама или экстремального программирования (XP), будучи раз внедренными, становятся очень «липкими». Почему? Во-первых, они быстро приносят реальную отдачу: у них привлекательное соотношение затрат и выгод и их внедрение быстро окупается. Во-вторых, эти практики не требуют мучительных усилий; большинство людей, которые их используют, говорит, что применяет их с интересом и даже с удовольствием (и, разумеется, с пользой). Люди есть люди: они предпочитают делать то, что им нравится.

Удобство и простота инструментов особенно важны в крупномасштабных разработках, так как *устойчивое внедрение* практик и процессов становится тем труднее, чем больше размер группы. Ваши инструменты должны *привлекать* людей, как яркие ароматные цветы притягивают к себе пчел.

Диаграммы причинно-следственных циклов

В этой книге мы будем не раз обращаться к диаграммам причинно-следственных циклов, потому что они помогают увидеть динамику того, что происходит при крупномасштабной разработке в больших группах. Уже только по одной этой причине полезно разобраться в этих диаграммах. Чтобы они были еще полезнее, мы рекомендуем:

Попробуйте… чертить диаграммы причинно-следственных циклов на групповых встречах, чтобы увидеть системную динамику.

Попробуйте… чертить диаграммы причинно-следственных циклов на доске вместе с коллегами

Практическая ценность этого совета намного важнее, чем может показаться на первый взгляд. Фраза «нам нужно системное мышление» слишком расплывчата и малоэффективна. Но, если вы со своими коллегами сделаете своей привычкой вместе вставать у большой доски и рисовать диаграммы причинно-следственных связей, это будет уже конкретная практика, которая превратит абстрактный призыв к системному мышлению в реальный системный подход – с потенциально далекоидущими последствиями.

Приведенные далее примеры могут показаться «теоретическими», когда они изложены на бумаге. Но представьте, что эти диаграммы были составлены вами в процессе живого обще-

ния с коллегами, когда вы вместе стояли у доски и горячо обсуждали каждый момент. Именно так мы рекомендуем применять системное мышление на практике.

Конкретный совет по построению диаграмм. Мы начинаем с того, что записываем на стикерах задействованные *переменные,* например «скорость разработки фич» или «количество дефектов», и размещаем стикеры на доске. Затем проводим между стикерами линии причинно-следственных связей. В процессе построения модели всегда приходится много переписывать, стирать, перерисовывать. Самый важный результат таких сессий – *совместное понимание*. Некоторые участники захотят сфотографировать то, что получилось на доске.

Нотации и примеры

Диаграммы причинно-следственных циклов содержат много элементов; вот список наиболее типичных и полезных, которые будут рассмотрены дальше в сценарии:

- переменные;
- причинно-следственные связи;
- обратные эффекты;
- ограничения;
- цели;
- реакции;
- «быстрые решения»;
- эффекты взаимодействия;
- сильные эффекты;
- отложенные эффекты;
- петли положительной обратной связи.

Далее приведен упрощенный сценарий разработки диаграммы для конкретной организации. Не рассматривайте это как общий случай.

Переменные. Диаграммы причинно-следственных циклов включают *переменные* (или метрики), такие как *скорость разработки (поставки) новой функциональности* и *количество дефектов*. Переменные имеют измеримую величину.

Причинно-следственные связи. Один элемент может влиять на другой, например увеличение скорости разработки может вести к увеличению количества дефектов; то есть чем больше объем нового кода, тем больше дефектов.

Скорость разработки

Уже на этом этапе можно столкнуться с *законом Вайнберга–Брукса* и *ловушкой причинно-следственной связи*. Нарисовать схему легко; совсем другое дело – построить ее с правильным пониманием. Например, рассмотрим взаимосвязь между *количеством разработчиков* и *скоростью разработки*.

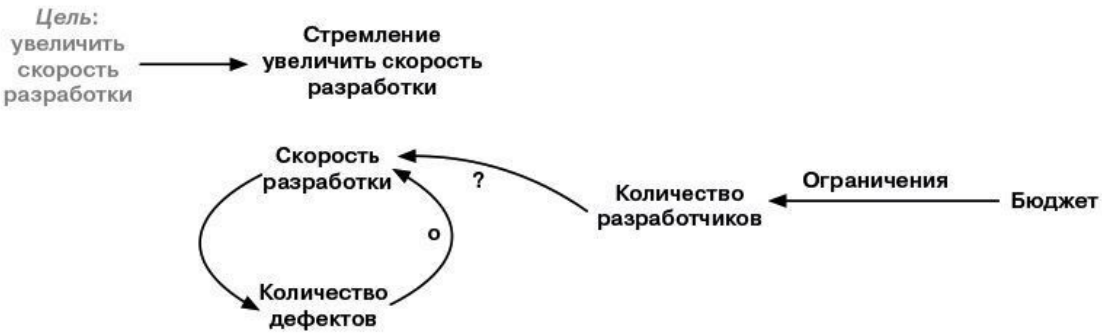
Природа некоторых причинно-следственных связей не всегда очевидна. Но людям свойственно делать поспешные выводы, что рост числа разработчиков ведет к ускорению разработки. Однако увеличение команды, особенно на поздних стадиях разработки, способно, наоборот, *уменьшить* скорость (подпункт «Закона Брукса» [Brooks95]). Кроме того, большое число плохих программистов может существенно замедлить работу, и часто бывает, что их *удаление* из группы *хорошо сказывается* на общей скорости.

Обратные эффекты. Эффект причинно-следственной связи может быть не только прямым, но и обратным: например, рост А ведет к росту Б, и наоборот. Обратный эффект обозначен на диаграмме символом «О» рядом с линией. Допустим, что увеличение количества дефектов создает препятствия для дальнейшей разработки, что, как следствие, замедляет скорость поставки новой функциональности, потому что люди тратят больше времени на исправление ошибок или поиск обходных путей.

Ограничения. Если только вам не удастся найти людей, готовых работать бесплатно, у вас есть ограничение на количество разработчиков, которых вы можете нанять исходя из имеющегося бюджета.

Ограничения – это *не* причинно-следственные связи. Увеличение бюджета не ведет напрямую к увеличению числа разработчиков.

Цели и реакции. Люди, группы и системы имеют цели – например, *увеличить скорость разработки*. Цели создают давление, побуждая людей реагировать (действовать) с намерением достичь этих целей. Но, принимая во внимание *закон Вайнберга–Брукса* и *ловушку причинно-следственной связи*, следует быть осторожными в своих предположениях о том, какие действия могут пойти на пользу в данной ситуации. Итак, добавим в нашу схему цель и реакции, которые она запускает:



Цель, подкрепленная *вознаграждением*, побуждает людей не только действовать, но и создавать *видимость* действий и достижений посредством **дисфункции измерения**, которая порождается предложенной системой вознаграждения. И дисфункция измерения может быть прямо пропорциональна воспринимаемой ценности вознаграждения, потому что люди мотивированы получить вознаграждение, а не улучшить систему [Austin96]. Давайте посмотрим, как вознаграждение может привести к снижению производительности системы. Системная динамика может выглядеть так:

Интересно, что вся эта системная динамика возникла из-за введения вознаграждения, причем между верхней частью модели и ее нижней частью пока нет явной взаимосвязи.

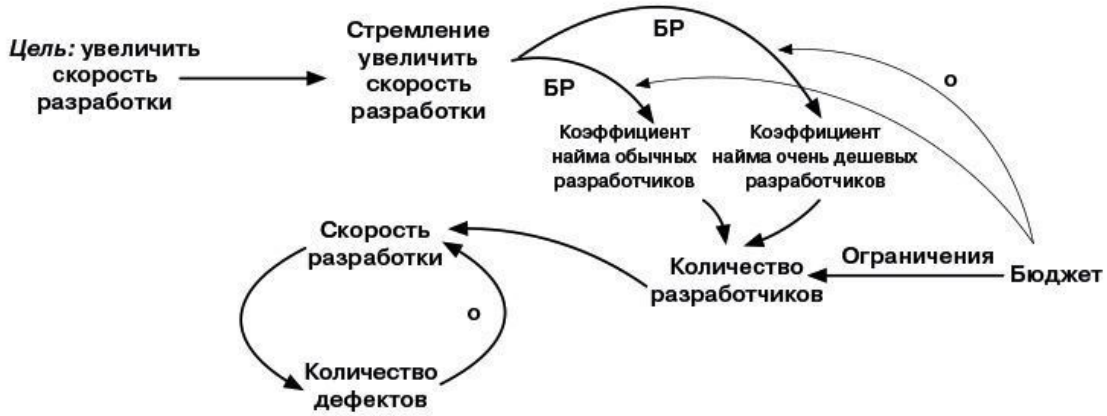
Словом, нет никаких гарантий того, что введение вознаграждения обусловливает увеличение скорости разработки – или даже влияет на нее.

Отказ от такой системы вознаграждения – фундаментальное решение, которое позволяет устранить корневую причину этой дисфункции. Другая, пусть более поверхностная, но тем не менее работоспособная контрмера – использование управленческой командой принципа бережливого менеджмента *«пойди и посмотри»*:

«Быстрые решения». Одно из трудных и медленных решений для увеличения скорости разработки – улучшить человеческий ресурс: нанять сильных разработчиков, расширить обучение и коучинг имеющейся команды, уволить слабых сотрудников. Альтернатива – достичь цели быстро и с наименьшими усилиями за счет «быстрого решения» или так называемого квик-фикса (*quick fix*). Иногда быстрое решение работает как в краткосрочной, так и в долгосрочной перспективе и действительно улучшает систему. Но иногда это не удается, и тогда получается как в поговорке: «Быстрее – значит медленнее». Например, если стоит цель увеличить скорость поставки новой функциональности, первое, что обычно приходит в голову людям, – увеличить количество разработчиков. Согласно их *предположению*, это самый быстрый и простой способ решить проблему со скоростью разработки. Итак, добавим в нашу схему это «быстрое решение» (символ БР):

Эффекты взаимодействия. Бюджет ограничивает наем новых сотрудников. Одно из трудных и медленных решений – найти способы увеличить бюджет. Быстрое и простое решение – нанять *намного более* дешевых разработчиков. В этом случае бюджет оказывает *эффект взаимодействия, влияя* на другие причинно-следственные связи. Когда принимается решение нанять дополнительных разработчиков, недостаточный бюджет вынуждает вас нанимать наиболее дешевых.

Просто провести (обратную) причинно-следственную связь напрямую от *бюджета* к *коэффициенту найма очень дешевых разработчиков* не совсем правильно, так как, по сути, это будет означать, что уменьшение бюджета ведет к увеличению найма очень дешевых разработчиков. Мы же хотим показать эффект взаимодействия – а именно то, что причинно-следственная связь А *влияет на причинно-следственную связь* Б. Чтобы графически показать это, мы проводим линию обратной причинно-следственной связи от *бюджета* к линии *быстрого решения* (БР), которая в результате раздваивается на *коэффициент найма обычных* и *очень дешевых разработчиков*:

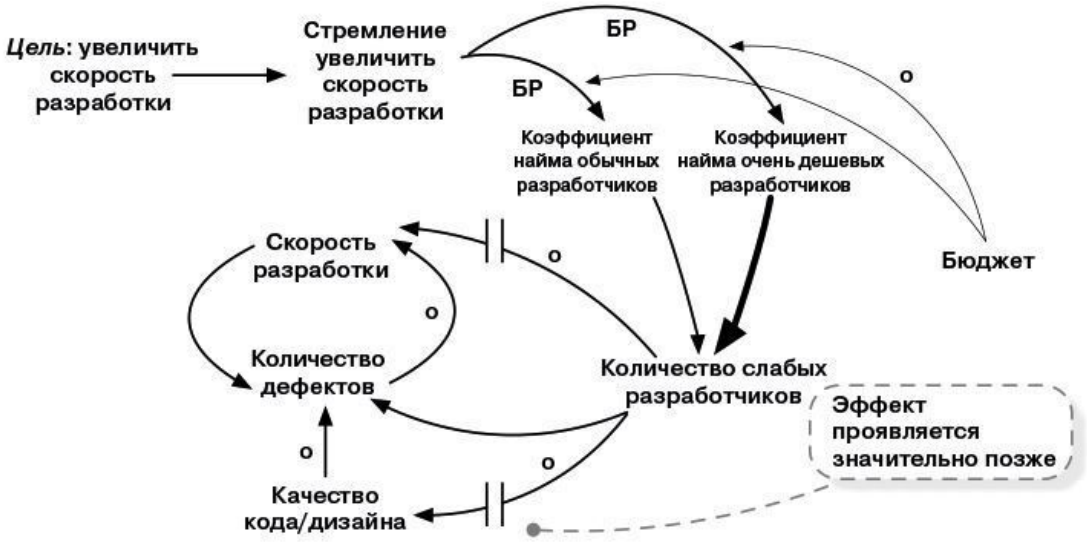


Сильные эффекты. Нам доводилось работать и с очень крутыми, но недорогими разработчиками, и с очень дорогими, но ужасными. Но, как правило, вы получаете то, за что платите, – нанимая разработчиков из большого пула дешевой рабочей силы, вы получаете в среднем гораздо более низкий уровень квалификации, чем при найме из пула дорогих. В нашей модели мы хотим показать, что наем очень дешевых разработчиков *резко увеличивает долю слабых (низкоквалифицированных) разработчиков* в группе. Чтобы показать такой *сильный эффект* в модели, мы используем толстую линию:

Отложенные эффекты. Одной из проблем при найме разработчиков является *заблуждение о небольшом разбросе в их квалификации* – ошибочное мнение, будто программисты не слишком отличаются друг от друга (с точки зрения производительности, качества кода и т. д.). Но исследования показывают, что верхний квартиль по уровню квалификации способен работать примерно вчетверо быстрее, чем нижний [Prechelt00]. Довольно большая разница, согласитесь. Кроме того, модель COCOMO, основанная на многолетних масштабных исследованиях, показывает, что уровень квалификации группы разработчиков – наиболее важный из всех факторов, влияющих на ее производительность [Boehm00]. Наконец, очень слабые программисты в среднем создают код худшего качества (с плохим дизайном) и с бóльшим количеством дефектов, что дополнительно тормозит всю систему.

Но все эти влияния проявляются не немедленно, а с некоторым запозданием. Например, если вы наймете много слабых разработчиков, пройдет относительно много времени, прежде чем начнут ощущаться последствия их плохой работы / некачественного кода. Соответственно среднее *снижение* скорости поставки новой функциональности (вызванное сильным влиянием разброса квалификации программистов) произойдет не сразу, а спустя какое-то время.

Чтобы показать такое *отложенное воздействие*, мы используем на модели линию с двойной чертой:



Отложенный характер эффектов негативно влияет на способность системы к *обучению* и коррекции. Если результат или случайное следствие какого-либо действия проявляется с большой задержкой во времени, люди часто не видят (причинно-следственной) связи между ними, не понимают, что именно А повлияло на Б или, еще сложнее, что А повлияло на Б, а Б в ответ повлияло на А.

Следовательно, люди не учатся и не исправляют ошибки – в правилах, управленческих действиях, инструментах и т. д. Именно из-за отложенных эффектов постепенное улучшение через практику бережливого подхода *кайдзен* может занимать длительное время: чтобы увидеть, улучшается ли что-то и как, требуется терпение и способность проникать в суть вещей.

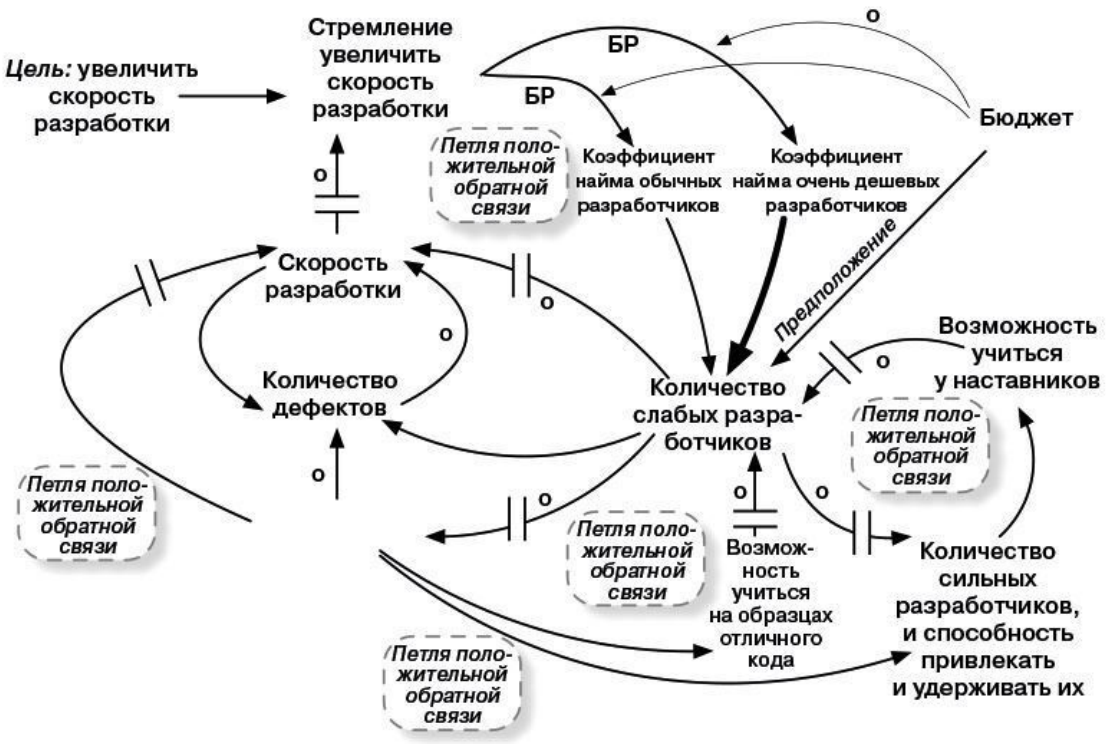
Петли положительной обратной связи. Петли положительной или отрицательной обратной связи[9] и отложенные эффекты – тонкие моменты, которые делают динамику системы еще более сложной и менее понятной. К примеру, как программисты могут повысить свой уровень квалификации? Один из способов – учиться у высококвалифицированных специалистов и видеть много примеров отличного кода. Но компания, в которой работает много (очень дешевых) программистов с низкой квалификацией, производит мало образцов качественного кода, а также не привлекает и не может удержать крутых программистов, которые могли бы играть роль наставников. Те скорее найдут работу в другом месте.

Таким образом, группа разработки входит в самоусиливающуюся нисходящую спираль – последовательность *петель положительной обратной связи,* где каждая петля усиливает последующую. К счастью, этот нисходящий тренд ограничен бюджетом.

> *Попробуйте… увидеть в системе петли положительной обратной связи.*

По мере того как уходит все больше крутых разработчиков, которые могли бы создавать отличный код и обучать других, у слабых разработчиков становится все меньше возможностей для обучения. Процент слабых разработчиков растет, а качество кода и производительность падают все ниже. Код становится все более грязным, запутанным, с большим количеством повторений, что снижает способность группы быстро реализовывать новую функциональность. Падение скорости реализации новой функциональности вынуждает нанимать еще больше дешевых разработчиков. Короче, в системе создается множество самоусиливающихся петель положительной обратной связи.

[9] Иногда под «петлями обратной связи» в этой книге подразумевается обратная связь в обычном понимании, а не в смысле системной динамики.



Подсказка: чтобы выявить петли положительной обратной связи, найдите циклы *с четным количеством* причинно-следственных связей с обратным эффектом. В модели выше мы видим пять таких петель.

Заключение

Приведенный сценарий – это только пример. Диаграмма причинно-следственных циклов позволяет наглядно представить сложную и неявную динамику системы рабочей среды. Лучше всего составлять ее на доске вместе с группой.

Учимся видеть ментальные модели

Приведенная выше диаграмма отражает не только реальные причинно-следственные связи, но и предположения людей о таких связях – ментальные модели, которые могут быть ошибочными. Необходимо отметить, что на человеческое восприятие причин и следствий влияет временно́й интервал между ними (отложенный эффект) и качество обратной связи в системе.

Цель выявления ментальных моделей в том, чтобы улучшить наш метакогнитивный навык видеть и подвергать сомнению собственные предположения и логические цепочки. Не ошибаемся ли мы в нашей логической схеме? Другая цель в том, чтобы узнать и обсудить (не пытаясь оскорбить чужую точку зрения) ментальные модели наших коллег.

Попробуйте… выявлять ментальные модели и предположения через совместную разработку диаграмм причинно-следственных циклов.

Увидеть ментальные модели – только первый шаг; второй, гораздо более трудный шаг – изменить их. Это искусство не входит в тематику данной книги, хотя успешное внедрение масштабированного скрама требует изменений в привычном образе мышления и выработки совместного понимания в большой группе.

Совет: чтобы выявить ментальные модели (предположения, убеждения, цепочки умозаключений и т. д.), связанные с системной динамикой, в ходе построения модели задавайте наводящие вопросы и записывайте ответы. Например: «Давайте поговорим о предположениях, стоящих за этой моделью. Что именно и почему заставляет нас так считать?»

Ответы записывайте на доске рядом с соответствующими элементами модели, например:

Пример: динамика «быстрее – значит медленнее»

Теперь, когда мы разобрались с тем, что такое «быстрые решения», отложенные эффекты, петли положительной обратной связи и ментальные модели, давайте рассмотрим интересную ситуацию, когда «быстрое решение» ведет сначала к кажущемуся краткосрочному улучшению какой-либо переменной, а затем к *отложенному ухудшению* той же переменной, – то есть динамику «быстрее – значит медленнее». Такая динамика часто встречается в рабочей среде и ведет к снижению производительности. Поэтому она заслуживает отдельной иллюстрации.

История Microsoft Word и ***секретного инструментария разработчика*** – классический пример динамики краткосрочного «улучшения» с последующим долгосрочным ухудшением. Речь идет о первом релизе Microsoft Word для Windows [Spolsky04]. Программа была выпущена на несколько *лет* позже, чем ожидалось. Почему? Потому что *менеджеры старались соблюсти первоначально составленный график и давили на разработчиков, чтобы уложиться в срок.*

Эта история показывает, почему склонность *принимать желаемое за действительное* рассматривается в бережливом подходе как один из факторов потерь. В нашем случае принятие желаемого за действительное выражалось в попытке строго следовать первоначальному графику, в основе чего лежало неправильное предположение (движимое все тем же соблазном принимать желаемое за действительное), будто исходные оценки проекта являются не оценками, а обязательствами, – распространенное заблуждение, которое часто ведет к деградации системы.

Рис. 2.2. иллюстрирует *базовую* динамику того, что происходит, когда менеджеры давят на людей с целью соблюсти первоначальный график, и почему такое «быстрое решение» проблемы медленного прогресса в краткосрочной перспективе вроде бы ускорило работу команды, но в долгосрочной – *замедлило* ее еще больше. В этой модели мы частично опустили более глубокую динамику, которая будет рассмотрена дальше и представлена на рис. 2.3.

Рис. 2.2. Динамика, вызванная давлением сроков и использованием секретного инструментария

Итак, в качестве «быстрого решения» менеджеры Microsoft уговаривали, подкупали (обещанием премий) и угрожали разработчикам Word, чтобы те уложились в первоначальный график. В результате разработчики вполне предсказуемо прибегли к своему **секретному инструментарию** – множеству практик, известных как «грязные хаки», которые ведут к написанию грязного кода (без тестов, без инспекции, с игнорированием известных дефектов, копипастом, плохим дизайном и т. д.), чтобы быстрее выпускать новую функциональность. Как видите, у разработчиков тоже есть «быстрые решения» своих проблем.

Поначалу казалось, что эта тактика обладает магическим действием. Чем настойчивее менеджеры давили на разработчиков, тем активнее те использовали свой секретный инструментарий и тем быстрее выпускались «фичи», что усиливало убеждение менеджеров, будто давление на разработчиков помогает. Но у этого обманчивого ускорения имелся отложенный эффект в виде долгосрочного замедления разработки (мы рассмотрим это чуть дальше). Развитие негативной динамики первое время оставалось незамеченным, поскольку последствия использования секретного инструментария проявились не сразу и руководство считало, что менеджерам не нужно самим быть опытными разработчиками, чтобы разбираться в деталях исходного кода или же регулярно проверять его состояние.

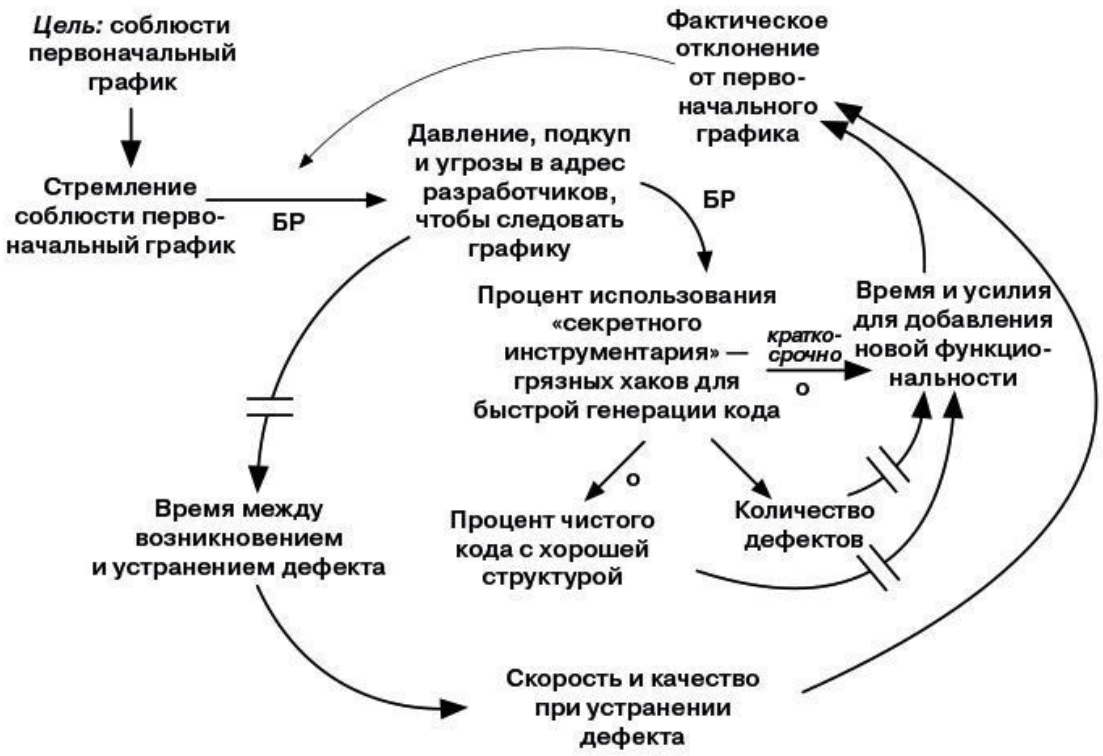


Рис. 2.3. Более глубокая динамика, вызванная давлением сроков иПроцент чиспользованием секретного инструментария

Более глубокий анализ системной динамики, который показывает, по какой причине разработка замедлилась в долгосрочной перспективе и почему первая версия Word для Windows была выпущена на несколько лет позже ожидаемого срока, представлен на рис. 2.3.

Естественно, значительный объем грязного кода в конечном итоге замедлил разработку. Хуже того, разработчики *игнорировали* постоянно растущий список открытых дефектов, генерируя вместо этого все больше новой функциональности. Это вело к тому, что между моментом возникновения дефекта и его устранением проходило все больше времени. Это в свою очередь значительно увеличило вариативность и время/усилия на устранение дефекта из-за усугубляющего негативного воздействия долгоживущей ошибки (например, из-за обходных решений и сильной связанности), а также потому что разработчики со временем забывали детальный контекст связанного с дефектом кода и, следовательно, им требовалось восстановить в памяти этот контекст, вокруг которого к тому времени оказалось еще больше грязного запутанного кода.

Проницательный читатель мог заметить несколько петель положительной обратной связи, которые усиливали этот цикл деградации; неудивительно, что при такой динамике продукт был выпущен на несколько лет позже, чем предполагалось.

Решение? Использовать принципы бережливого подхода «*остановись и исправь*» и «*пойди и посмотри*». Во-первых, когда возникают проблемы, вместо того чтобы пытаться двигаться быстрее, менеджер-учитель побуждает людей *замедлиться,* понять системную динамику и корневые причины и устранить их – чтобы улучшить *систему*. Двигаясь медленнее, Toyota – чемпион бережливого мышления – стала одной из самых быстро развивающихся компаний в мире. Во-вторых, менеджерам нужно *пойти туда, где делается реальная работа, и посмотреть, что происходит*. «Реальная работа» в программировании – это код; следовательно, менеджеры первого уровня должны быть квалифицированными программистами, которые способны оценить качество кода и должны делать это регулярно.

Когда релиз все-таки состоялся, люди в Microsoft наконец-то провели ретроспективу и проанализировали произошедшее. Результатом стало принятие компанией политики «*ноль дефектов*», которая требует в первую очередь исправить все известные дефекты в разрабатываемом коде – свести список открытых дефектов к нулю – и только затем приступать к работе над новой функциональностью.

Учимся видеть корневые причины

«Мы пытались внедрить скрам на протяжении всего прошлого года, но не увидели больших улучшений. Почему?» Выявление корневых причин может помочь найти ответ. Системное мышление требует от нас развивать умение «проникать в суть» – видеть фундаментальные причины и более глубокие действующие силы. Непредвиденные последствия и «быстрые решения» – это симптомы того, что люди не понимают сути происходящего.

Попробуйте… выявлять корневые причины с помощью причинно-следственного моделирования, метода «Пять почему» и диаграмм Исикавы

Непрерывное улучшение – один из двух столпов бережливого подхода. В Toyota существует культура «останови и исправь», которая состоит в следующем:

1. Когда люди видят проблему, они останавливаются…
2. Анализируют ее, чтобы *найти корневые причины*…
3. Проводят эксперименты по изменению процесса, чтобы исправить проблему и улучшить систему.

Следующие простые инструменты помогают обсудить проблему и выявить корневые причины:

- метод **«Пять почему»** (рассматривается в главе о *бережливом подходе);*
- **диаграммы Исикавы** («Рыбья кость»).

Оба инструмента следует применять на групповых встречах – например, в ходе ретроспектив спринта в скраме.

Рис. 2.4. Схема «Пять почему», нарисованная в ходе ретроспективы спринта

Выявление корневых причин с помощью диаграмм Исикавы

Метод «Пять почему» относительно неструктурирован; его можно комбинировать с **диаграммами Исикавы** (диаграммами «Рыбья кость») [Ishikawa86], чтобы организовать и свя-

зать причины, лежащие в основе проблемы, например такой, как *неэффективность скрам-мастеров*. Шаг первый – провести мозговой штурм по причинам проблемы; мы рекомендуем *брейнрайтинг*, когда каждый участник записывает свои идеи по одной на листочке бумаги и кладет эти листочки на общий стол. Шаг второй – *кластеризация по сходству*; нужно сгруппировать причины в семейства связанных причин и дать название каждой группе (рис. 2.5). Шаг третий – нарисовать скелет диаграммы Исикавы, где в качестве «костей» используются названия групп. Шаг четвертый – применить метод «Пять почему» к каждой группе или к заслуживающим внимания подэлементам каждой группы, чтобы выявить корневые причины. Все ответы нужно заносить в диаграмму, как показано на рис. 2.6.

Рис. 2.5. Брейнрайтинг и кластеризация по сходству, Valtech India

Рис. 2.6. Диаграмма Исикавы, построенная с помощью брейнрайтинга и кластеризации по сходству и углубленная с помощью метода «Пять почему»

Рис. 2.7. Использование диаграммы Исикавы для анализа корневых причин на групповой встрече

После анализа корневых причин: проведите корректирующие эксперименты

Анализ корневых причин проводится не ради любопытства. Когда вы узнаете первопричины проблемы, следующий важный шаг, который необходимо сделать в рамках ретроспективы спринта, – провести эксперименты по улучшению системы [Vodde07].

Учимся видеть (и слышать) локальную оптимизацию

«Каждый старается изо всех сил, но производительность всей системы продолжает ухудшаться. Как такое может быть?» Это парадокс **локальной оптимизации** – когда принимающие решения лица стремятся оптимизировать систему исходя из локальной точки зрения или из собственных интересов (индивидуальных либо на уровне отдела). При этом человек обычно *убежден, что принимает наилучшее решение*, но, поскольку это «наилучшее» решение является локальной оптимизацией, на деле оно ведет к субоптимизации общей производительности системы. Среди причин такого поведения – «бункерное мышление», недопонимание, страх, ограниченная информированность, плохая обратная связь, невежество, карьеризм, жадность и прочие распространенные *изъяны организационного обучения*.

> *Попробуйте… увидеть (и услышать) случаи локальной оптимизации; это бич больших продуктовых групп.*

У небольшой продуктовой группы из 30 человек нет ни времени, ни сил заниматься чем-то подобным. Но большие компании с большими продуктовыми командами, централизованными процессами и инструментами, центральным «проектным офисом» и т. д., кажется, возвели локальную оптимизацию и связанные с ней потери в ранг искусства. Вершина этого искусства, без сомнения, государственная бюрократия. Следовательно, когда вы внедряете скрам на больших масштабах, умение *видеть (и слышать)* локальную оптимизацию и справляться с ней становится особенно важным.

К примеру, юридический отдел и отдел корпоративной безопасности вводят чрезвычайно важное, на их взгляд, правило: в целях предотвращения утечки интеллектуальной собственности они решают «запретить размещение на стенах любой информации». Или же хозяйственный отдел в соответствии с политикой снижения затрат требует «содержать все стены в чистоте и порядке». Таким образом, они фактически запрещают практику визуального управления, принятую в бережливом подходе (для которой обычно используются стены), и препятствуют другой хорошо себя зарекомендовавшей практике – групповой работе у доски. Возможно, юристы с помощью этой меры действительно снизят риск потери интеллектуальной собственности (хотя это еще под вопросом), а хозяйственный отдел будет доволен чистыми стенами – ценой подавления инноваций и совместной работы в группе разработки. В конечном итоге такое замедление инноваций и скорости разработки продуктов приведет к тому, что у компании будет все меньше ценной интеллектуальной собственности, которую нужно защищать. Зато юристы успешно выполнят распоряжение топ-менеджмента «обеспечить защиту нашей интеллектуальной собственности», а хозяйственники – «обеспечить чистый офис». Они просто *стараются как можно лучше делать свою работу*.

Ниже приведен фрагмент из реального электронного письма «офисной полиции» одной компании с запретом использовать визуальные инструменты управления на стенах. Подумайте, какова цель этой локальной оптимизации и какие ментальные модели могут лежать в ее основе?

> *Индивидуальную рабочую зону в кубиклах можно персонализировать. При этом ничто не должно превышать уровень перегородок или нарушать гармонию офисного пространства.*

Склонностью к локальной оптимизации нередко страдают и централизованные группы, которые выбирают программное обеспечение для других. Как правило, они стараются найти такой инструмент, который максимально снижает предполагаемые затраты (любопытно, но такие группы почему-то редко выбирают бесплатные инструменты с открытым исходным кодом) или лучше всего подходит для выполнения определенных специфических задач или

для какой-то конкретной рабочей роли (хотя пользоваться этим инструментом придется *всем*), вместо того чтобы максимизировать общую скорость и способность системы поставлять ценность клиентам.

Большинство случаев локальной оптимизации могут рассматриваться как варианты нарушения принципа «*Следи за эстафетной палочкой, а не за бегунами*».

При масштабном внедрении скрама или принципов гибкой разработки большинство возникающих вопросов типа «Да, но?..» касаются именно локальной оптимизации, например: *«Да, но… как насчет управленческой отчетности?»* или, более обобщенно, *«Да, но… как насчет <этого конкретного случая>?»*. В результате правила и практики меняются таким образом, чтобы удовлетворить требования отчетности или служить любой другой второстепенной цели, вместо того чтобы обеспечивать *глобальную оптимизацию* – улучшать общую способность системы быстро создавать ценность. Иногда можно встретить *локальную оптимизацию ради редких или чрезвычайных случаев*. Например, группа, отвечающая за централизованный выбор программного инструмента для всей компании, фокусируется на самых сложных и редких случаях его использования и на основе этого сценария выбирает инструмент, который подходит именно для таких случаев. В результате достигается субоптимизация для 5 % случаев – за счет простоты использования и скорости в 95 % случаев.

Еще одна причина локальной оптимизации – консервативное мышление и незнание новых подходов к работе. Особенно часто это встречается в крупных продуктовых группах. Например, однажды мы помогали большой сетевой продуктовой группе в Европе внедрить скрам и практику *непрерывной интеграции* (НИ) вместе с системой НИ, которая постоянно интегрировала, собирала и автоматически тестировала продукт. Спустя некоторое время внешний менеджер проверил, как идут дела, и потребовал изменить практику интеграции, потому что у группы не было формального плана, в соответствии с которым менеджер по интеграции должен был вручную интегрировать все ПО, и, разумеется, не было и самого менеджера по интеграции. Внешний менеджер хотел «оптимизировать» работу менеджера по интеграции, который был больше ни к чему. Он не понимал, что вся эта устаревшая модель работы стала не нужна благодаря НИ. Эта история повторяется во всех крупных устоявшихся компаниях-разработчиках, где локальной оптимизации подвергаются *существующие* традиционные способы работы, такие как ручное тестирование, или отдельные группы по разработке архитектуры, компонентные команды и т. д. Консультанту, который занимается внедрением масштабированного скрама на уровне компании, приходится разгребать *горы* таких локальных оптимизаций.

В бережливом подходе и гибкой методологии внимание всегда направлено на глобальные системные цели – создавать ценность быстро с высоким качеством и высоким моральным духом, то есть на **глобальную оптимизацию**. Старайтесь рассматривать все решения и правила через эту призму. Чтобы выработать культуру «оптимизации целого», всегда задавайте следующий вопрос:

> *Помогает ли это решение или правило быстрее поставлять ценность внешнему конечному потребителю или же оно нацелено на интересы конкретного отдела или человека, конкретную внутреннюю политику/практику или редкий случай?*

В скраме именно владелец продукта отвечает за выбор целей с максимальной ценностью, которые ведут к созданию потенциально готового к поставке продукта в конце каждого цикла, максимизируют отдачу от инвестиций и позволяют превзойти ожидания клиента, при этом поддерживая устойчивый темп работы и высокое качество разработки. В этом и состоит суть скрама – переориентировать систему с локальной на глобальную оптимизацию.

Заключение

Овладейте системным мышлением сами и поощряйте к этому других. Мы советуем вам собираться с коллегами у доски и рисовать диаграммы причинно-следственных циклов и другие системные модели, чтобы превратить системное мышление в реальный рабочий инструмент.

Рекомендуемая литература

■ «Выход из кризиса: Новая парадигма управления людьми системами и процессами» Уильяма Эдвардса Деминга (М.: Альпина Паблишер, 2022) – основополагающий труд одного из самых выдающихся системных мыслителей и экспертов по качеству. Книга начинается с заявления скромной цели: «Цель этой книги – преобразование американского стиля менеджмента. Это не означает его перестройку или ревизию. Преобразование требует полностью обновленной структуры сверху донизу». Для этого, по мнению Деминга, менеджерам необходимо владеть *системой глубинных знаний*, то есть: (1) понимать, что такое *система*; (2) понимать вариабельность, обусловленную общими или особыми причинами (с этим, в частности, имеет дело теория массового обслуживания); (3) признавать ограниченность знаний и быть готовыми обсуждать ошибки и (4) обладать современными знаниями в области психологии и социологии, чтобы не руководить поведением и мотивацией людей только лишь на основе «здравого смысла». Книга построена вокруг знаменитых 14 принципов управления, один из них: *«Исключите управление по целям. Перестаньте управлять по числам и количественным результатам. Замените его лидерством».*

■ «Мировая динамика» Джея Форрестера (М.: АСТ, 2003) – классическая работа по системной динамике, ясная и содержательная. Написанная в начале 1960-х гг., она не теряет своей актуальности и сегодня. Наряду с моделированием причинно-следственных связей в работе рассматривается моделирование потоков и запасов – информационных, денежных, материальных – в системах. Также затрагивается тема формального математического моделирования, но последнее необязательно для понимания системной динамики.

■ «Управление качеством ПО: Системное мышление» (Quality Software Management: Systems Thinking) и «Введение в общее системное мышление» (An Introduction to General Systems Thinking) Джеральда Вайнберга, безусловно, достойны прочтения. Написаны опытным консультантом по развитию систем.

■ «Пятая дисциплина: Искусство и практика обучающейся организации» Питера Сенге (М.: МИФ, 2018) – еще один классический труд, который доказывает необходимость применения лидерами системного мышления (пятой дисциплины), а также других четырех ключевых дисциплин для создания сильной и устойчивой компании. Эти четыре дисциплины: 1) личное совершенствование; 2) умение критически смотреть на собственные убеждения и мышление (интеллектуальные модели) и видеть в них ошибки; 3) умение создавать и сообщать другим значимое общее видение и 4) способность команд к обучению. Мы рекомендуем – по крайней мере на старте – игнорировать концепцию «архетипов», представленную в этой книге. Хотя она задумывалась как вспомогательный обучающий инструмент, мы заметили, что она отвлекает и даже отпугивает людей на начальном этапе обучения и применения базового моделирования системной динамики. «Архетипы» не являются частью оригинального подхода к системной динамике.

Конец ознакомительного фрагмента.

Текст предоставлен ООО «ЛитРес».

Прочитайте эту книгу целиком, купив полную легальную версию на ЛитРес.

Безопасно оплатить книгу можно банковской картой Visa, MasterCard, Maestro, со счета мобильного телефона, с платежного терминала, в салоне МТС или Связной, через PayPal, WebMoney, Яндекс.Деньги, QIWI Кошелек, бонусными картами или другим удобным Вам способом.