Кілька слів про книжку Девіда Андерсона

Книжка Девіда із системи канбан відчутно вплинула на мій підхід до розробки ПЗ і змінила мій спосіб мислення про процеси. Замість того, аби розглядати роботу в термінах історій, балів і таймбоксів, віднині я чітко бачу *WIP*, потік і каденцію. Ця книжка вдало доводить зазначену проблематику до широкої аудиторії та є обов'язковою для тих, хто перебуває в пошуках шляхів успішного та сталого розвитку своєї організації.

Карл Скотланд, старший консультант «EMC Consulting»

Канбан є інтригуючою темою, оскільки його широке впровадження може бути реалізоване лише за умови врахування особливостей робочих процесів і перешкод, що виникають. Девіду вдалося закласти міцне теоретичне підґрунтя і при цьому суворо дотриматися принципів практичної доцільності та реалістичної оцінки результатів.

Кріс Сіммонс, менеджер із розвитку в компанії «Sophos»

Книжка Девіда Андерсона «Канбан» далеко виходить за межі звичайного знайомства із тими рушійними силами, що їх надає впровадження методу Канбан. Вона забезпечує чітке роз'яснення підвалин Канбан, надає влучні приклади та практичні поради. У сфері інтелектуальної праці Канбан сприяє прагненню до самостійності на робочому місці та є однією з найбільш видатних управлінських систем сучасності.

Крістин Скасків, Agile-коуч

Найкраща нова методологія змін у галузі розробки програмного забезпечення, яку я бачив за останнє десятиліття.

Девід А. Булкін, віцепрезидент «LitheSpeedLLC»

ЗМІСТ

Частина третя. УПРОВАДЖЕННЯ КАНБАН

Розділ 6. Складання карти потоку створення цінності

Розділ 7. Координація в канбан-системах

Розділ 8. Формування каденції поставки

Розділ 9. Формування каденції поповнення

Розділ 10. Налаштування лімітів *WIP*

Присвячується Ніколя та Наталі

• Передмова •

Я завжди звертаю увагу на роботу Девіда Андерсона. Познайомились ми в жовтні 2003 року, коли він надіслав мені примірник своєї книжки *Agile Management for Software Engineering: Applying Theory of Constraints for Business Results*. Як можна припустити з назви, на цю роботу відчутний вплив справила теорія обмежень Еліягу Голдратта (*TOC*). Пізніше, у березні 2005 року, я відвідав Девіда в *Microsoft*; на той час він здійснював вражаючу роботу з діаграмами накопичувального потоку. Згодом, у квітні 2007 року, мені довелося спостерігати, як діє проривна система канбан, запроваджена ним у компанії *Corbis*.

Усю цю хронологію я наводжу для того, аби ви відчули, у якому нестримному темпі просувається управлінське мислення Девіда. Він не тримається за одну-єдину ідею, намагаючись підігнати під неї весь світ. Натомість він намагається розглядати проблеми у якнайширшому плані, будучи відкритим для різних варіантів рішень, перевіряє винайдене на практиці і пояснює причини дієвості цього. Результати подібного підходу ви побачите в цій книжці.

Звісно, швидкість — річ корисна, якщо вона спрямована куди треба, а я впевнений, що Девід рухається в правильному напрямку. Особливе захоплення в мене викликає його остання робота з системами канбан. Я завжди вважав, що принципи ощадливого виробництва можна безпосередньо застосовувати в розробці програмних продуктів — на відміну від ідей *TOC*. Ба більше: ще в жовтні 2003 року я писав Девіду таке: «Однією з головних слабкостей *TOC* є недооцінка важливості розміру партії. Якщо головним пріоритетом є пошук та усунення обмежень, то зазвичай це означає, що ви просто вирішуєте не ту проблему». Я досі вважаю це твердження істинним.

Під час нашої зустрічі 2005 року я знову запропонував Девіду не задовольнятися фокусуванням на вузьких місцях у *TOC*. Я пояснював

йому, що гучний успіх виробничої системи компанії *Toyota* (*TPS*) не має нічого спільного з пошуком й усуненням вузьких місць. Збільшення продуктивності *Toyota* було досягнене за рахунок зменшення обсягів партії і варіативності, завдяки чому скоротився перелік незавершених робіт (*work-in-progress, WIP*). Саме скорочення цього переліку призвело до досягнення економічних переваг, і це стало можливим завдяки таким системам скорочення незавершеного виробництва, як канбан.

У 2007 році я відвідав компанію *Corbis*. Результат тамтешнього впровадження канбан справив на мене неабияке враження. Я зауважив Девіду, що він значно поліпшив канбан-підхід, що використовується в *Toyota*. Чому я так вирішив? Виробнича система *Toyota* оптимізована для роботи з повторюваними та передбачуваними завданнями з тією самою тривалістю і однорідною вартістю затримки. За таких умов цілком коректно використовувати такі підходи, як *FIFO*-пріоритизація (*first-in-first-out*, тобто «перший надійшов — перший обслуговується»). Також цілком доцільно блокувати надходження роботи, якщо досягнуто межі *WIP*. Однак у випадку, коли ми маємо справу з неповторюваною, непередбачуваною діяльністю з різною тривалістю завдань та різними вартостями затримки, ці підходи не можна визнати вдалими — а саме так стоять справи з розробкою ПЗ. Нам потрібні більш просунуті системи, і перед вами — перша книжка, що описує їх з численними практичними подробицями.

Мушу попередити читачів про деякі речі. По-перше, якщо ви думаєте, що знаєте, як працюють канбан-системи, то ви, мабуть, маєте на увазі ті з них, що використовуються в ощадливому виробництві. Ідеї, на яких базується книжка Девіда, виходять далеко за рамки тих простих систем, у яких використовуються статичні *WIP*-ліміти, *FIFO*-планування і єдиний клас обслуговування. Будь ласка, зверніть увагу на ці відмінності!

По-друге, не думайте, що цей підхід є системою візуального контролю. Візуалізація *WIP*, що досягається за допомогою канбан-дощок, дуже корисна, але це лише один з аспектів даного підходу. Якщо ви уважно прочитаєте цю книжку, то знайдете в ній багато цікавого. Найцінніша інформація приховується, наприклад, у таких аспектах, як структури процесів надходження та відправки завдань, управління незамінними ресурсами і використання класів обслуговування. Не відволікайтеся на візуальну частину, інакше проґавите найцікавіші моменти.

По-третє, не відкидайте ці методи лише тому, що вони здаються занадто простими в застосуванні. Простота в застосуванні є результатом ідей Девіда з отримання максимальної користі з мінімальними зусиллями. Він чітко усвідомлює потреби практиків і приділяє максимум уваги тому, що дійсно працює. Прості методи демонструють високу стабільність і майже завжди призводять до найкращих результатів у довгостроковому плані.

Це захоплива й важлива книжка, і вона безумовно заслуговує на ретельне вивчення. Рівень користі, яку ви отримаєте від неї, залежить від того, наскільки серйозно ви поставитеся до читання. Жодний інший посібник не познайомить вас із цими передовими ідеями краще. Сподіваюся, що «Канбан» Дейва Андерсона сподобається вам так само, як сподобався мені.

Дон Рейнертсен, автор книжки
«The Principles of Product Development Flow»

Частина перша

ВСТУП

• Розділ 1 •

Розв'язання дилеми *Agile*-менеджера

У 2002 році я був менеджером із розробки у віддаленому офісі підрозділу мобільних телефонів *Motorola PCS* у Сіетлі, штат Вашингтон. Мій відділ був частиною стартапу, який *Motorola* придбала роком раніше. Ми розробляли мережеве ПЗ для бездротової передачі даних на кшталт бездротового завантаження та управління приладами. Ці серверні застосунки були частиною інтегрованих систем, які працювали разом із клієнтськими застосунками на мобільних телефонах, а також з іншими елементами в телекомунікаційних мережах та операційній інфраструктурі, на кшталт білінгової системи. Дедлайни призначалися менеджерами, які не звертали уваги на інженерну складність проєкту, його ризики та обсяг. Наш код розвивався від початку стартапу, при цьому ми зрізали багато кутів. Один старший розробник наполягав на тому, щоб наш продукт називався прототипом. Ми мали відчайдушну потребу в підвищенні продуктивності та якості продукції, аби відповідати потребам бізнесу.

У власній повсякденній діяльності того року, а також під час роботи над моєю попередньою книжкою[1] мене турбували переважно два питання. Перше — яким чином захистити команду від постійно зростаючих запитів бізнесу і досягти того, що сьогодні в *Agile*-співтоваристві зветься сталим темпом? І друге — чи зможу я успішно запровадити *Agile*-підхід у масштабах всієї організації, долаючи неминучий опір змінам?

---

[1] Anderson, David J. *Agile Management for Software Engineering: Applying the Theory of Constraints for Business Results*. Prentice Hall, 2003.

Мій особистий пошук сталого темпу

У 2002 році *Agile*-спільнота сприймала сталий темп просто як «сорокагодинний робочий тиждень»[1]. *Agile*-маніфест, зі свого боку, зазначав: «*Agile*-процеси сприяють оптимальному розвитку. Спонсори, розробники і користувачі мають бути готові підтримувати постійний ритм протягом необмеженого часу»[2]. За два роки до цього моя команда в *Sprint PCS* повсякчасно чула від мене, що «розробка великих за обсягом програмних продуктів — це марафон, а не спринт». Якщо членам команди потрібно підтримувати сталий темп під час здійснення півторарічного проєкту, то дуже важливо завадити їхньому вигоранню за місяць-другий від початку роботи. Проєкт потрібно розпланувати, бюджетувати, розписати за часом і оцінити так, аби члени команди щодня витрачали на роботу розумну кількість часу і не надто втомлювалися. Переді мною як менеджером постало завдання досягти цієї мети і задовольнити всі вимоги бізнесу.

У 1991 році, коли я працював на своїй першій менеджерській посаді в стартапі зі створення плат відеозахоплення для ПК та більш дрібних комп'ютерів, *CEO* повідомив мені, що в керівництва склалося про мене «вкрай негативне враження». Причина? Я завжди відповідав «ні», коли від моєї команди вимагали все більше продуктів або функцій, а наші потужності як розробників уже були майже вичерпані. До 2002 року це стало моєю звичкою: понад десять років поспіль я відмовлявся виконувати незліченні примхи власників бізнесу.

Загалом команди розробників ПЗ і *IT*-відділи компаній неабияк залежали від інших груп, що повсякчас торгувалися, просили, погрожували і вимагали переробити навіть найбільш обґрунтовані та об'єктивно розроблені плани. До переліку вразливих потрапляли навіть ті плани, що ґрунтувалися на ретельному аналізі та багаторічному досвіді. Більшість же команд, які не володіли переконливими методами аналізу й певним досвідом, були безсилі перед тими, хто намагався примусити їх узяти на себе невідомі (і переважно нездійсненні) зобов'язання.

Отже, співробітники поступово змирилися з шаленим навантаженням, і надмірні обсяги роботи перетворилися на норму. Скидалося на те, що інженери-програмісти взагалі не потребують спілкування з друзями або сімейного життя. Звучить несправедливо, бо так воно і є! Мені відомі численні приклади, коли надмірне виробниче навантаження руйнувало сімейні стосунки.

[1] Kent, Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 2000.

[2] Beck, Kent,et al. *The Principles Behind the AgileManifesto* (http://www.agilemanifesto.org/principles.html).

Водночас неможливо було співчувати типовому розробнику ПЗ тих часів. У моєму рідному штаті Вашингтон інженери-програмісти посідали друге місце за рівнем річного прибутку після стоматологів. Як і за часів Генрі Форда, тобто в 1920-х роках, коли люди на його автомобільних заводах заробляли уп'ятеро більше, ніж у середньому по країні, нікому навіть на думку не спадало поскаржитися на монотонність роботи або відсутність дозвілля, оскільки їхня праця щедро оплачувалася. Важко уявити собі діяльність профспілок у таких інтелектуальних галузях, як розробка ПЗ, тому ніхто й не намагався всерйоз дослідити причини фізичних і психологічних недуг, від яких потерпають розробники. Більш відповідальні роботодавці згодні сплачувати додаткові медичні послуги на кшталт масажу або психотерапії, або надавати працівникам «дні психічного здоров'я» замість того, аби приділити увагу виявленню основних причин проблеми. Технічний письменник, співробітник відомої фірми-розробника ПЗ, якось сказав мені: «Немає нічого страшного в тому, що я вживаю антидепресанти, адже так роблять усі!» Стикаючись із такими зловживаннями, інженери-програмісти зазвичай погоджуються з усіма вимогами, отримують непогану зарплату й страждають від наслідків психологічного вигорання.

Я прагнув змінити такий стан справ, знайти взаємовигідний підхід, що дозволив би мені казати «так» керівництву й при цьому захищати свою команду, забезпечуючи досягнення сталого темпу. Я хотів повернути членам своєї команди нормальне життя і поліпшити умови, що викликали стреси та проблеми зі здоров'ям розробників, які не досягли навіть тридцяти років. Ось чому я вирішив розпочати боротьбу із цими проблемами.

Мій особистий пошук успішного управління змінами

Друга проблема, якою я переймався,— це прагнення кинути виклик великим організаціям для здійснення змін в управлінні. Я був менеджером із розробки в *Sprint PCS* і згодом у *Motorola*. У цих компаніях існувала нагальна потреба в переході на більш гнучкі методи роботи. Але в обох випадках було складно застосовувати *Agile*-методи більш як до однієї-двох команд.

Обидва рази я не мав достатньо повноважень, щоби просто наказати впровадити зміни до роботи численних команд. Я намагався ввести зміни на прохання вищого керівництва, але не володів належними повноваженнями. Мене просили вплинути на колег, аби ті запроваджували в своїх командах такі ж зміни, як і я у власній команді. Однак інші команди опиралися впровадженню методів, які, на мій погляд, якнайкраще зарекомендували себе в моєму випадку. Ця протидія, імовірно, мала кілька причин, але найчастіше я чув, що в кожної команди власна ситуація, і мої методи слід ретельно підганяти під конкретні потреби інших. У середині 2002 року я зрозумів, що примусове виконання наказів стосовно будь-якого процесу розробки ПЗ не має сенсу — це просто не спрацює.

Процес дійсно потребував адаптації для кожної конкретної ситуації. А також вимагав активного керівництва кожною з команд. Проте саме його часто не вистачало. Навіть за умови належного керівництва я не мав цілковитої впевненості, що суттєві зміни можуть відбутися без відповідного фрейморку, а також без принципів та рекомендацій щодо того, як адаптувати процеси для будь-якої ситуації. Якщо керівнику, коучу або інженеру-технологу бракує чіткого уявлення про те, що робити, будь-яка адаптація, найвірогідніше, відбудеться на його власний розсуд, ґрунтуючись на його власних переконаннях та забобонах. Це з тією ж імовірністю викличе хвилю нарікань і заперечень, як і використання недоречних шаблонних процесів.

Я спробував висвітлити цю проблему в книжці *Agile Management for Software Engineering*, над створенням якої працював саме тоді. Я ставив питання: «Чому гнучка розробка дає кращі економічні результати, ніж традиційні підходи?» і прагнув застосувати в пошуках відповіді структуру теорії обмежень[1].

Під час досліджень і створення тексту я зрозумів, що кожна ситуація певною мірою унікальна. Хіба ж може той чи інший стримуючий фактор або вузьке місце виявитися однаковим для будь-якої команди і проєкту в будь-який час? Кожна команда унікальна: різні навички, можливості, досвід. Кожен проєкт відрізняється від інших бюджетом, обсягом і ступенем ризиків. Не схожі одна на одну й організації: у них різні ланцюжки створення цінності, вони працюють на різних ринках

[1] *Див.:* Goldratt, Eliyahu M. *What is this thing called The Theory of Constraints and How should it be implemented?* North River Press, 1999.

(рис. 1.1). Мені здалося, що це може пояснити причини виникнення опору змінам. Якщо запропоновані зміни методів роботи і моделей поведінки не дають, на думку членів команди, очевидної переваги, то вони не сприймають їх. Якщо ці зміни не впливають на те, що сприймається командою як обмеження або стримуючий фактор, то команда намагається чинити опір. Інакше кажучи, зміни, запропоновані без урахування контексту, будуть відкинуті людьми, які краще за всіх знають контекст проєкту.

Команди мають різні...	**Проєкти мають різні...**
• Набори навичок • Рівні досвіду • Рівні можливостей	• Бюджети • Розклади • Обсяги робіт • Ступені ризику
Організації мають різні... • Ланцюжки створення цінності • Цільові ринки	

Рис. 1.1. Чому універсальна методологія розробки не працює

Здавалося б, буде краще, якщо новий процес почне розвиватися, усуваючи одне обмеження за іншим. Це основна теза теорії обмежень Ґолдратта. Розуміючи, що мені ще багато чому треба навчитися, я, однак, усвідомлюючи цінність зібраного матеріалу, поринув у роботу над попередньо розпланованим рукописом. Я чудово розумів, що книжка не допоможе впровадити ці ідеї у більшому маштабі, оскільки майже зовсім не дає порад з управління змінами.

Підхід Ґолдратта[1] був спрямований на виявлення обмежень, а згодом і способів їх усунення, аби вони припинили негативно впливати на продуктивність. Після цього виникають нові обмеження, і цикл повторюється. Це ітеративний підхід, що передбачає систематичне поліпшення продуктивності шляхом виявлення та усунення перешкод.

Нарешті я зрозумів, що можна поєднати цей підхід із деякими прийомами зі сфери ощадливого виробництва. Змоделювавши робочий процес розробки ПЗ у вигляді потоку створення цінності, а потім створивши систему відстеження і візуалізації стану робіт відповідно

[1] Див. Розділ 16.