Математические структуры и моделирование 2017. № 3(43). С. 96–107

УДК 004.053
DOI: 10.25513/2222-8772.2017.3.96-107

О ФОРМАЛИЗАЦИИ ПРОЦЕССА РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Е.А. Тюменцев
генеральный директор, e-mail: etyumentcev@hwdtech.ru

LLC Hello World! Technologies

Аннотация. В книге «Мифический человеко-месяц, или Как создаются программные системы» Брукс ссылается на несколько исследований, на основании которых можно сделать вывод, что производительность труда программистов, измеряемая в количестве строк кода в единицу времени, падает по мере роста размера программного проекта. В настоящей работе описывается формализация процесса разработки программного обеспечения, как процесса редактирования исходного текста программы, с помощью которой определяется функция трудоёмкости, а также выводится достаточное условие постоянной производительности труда программистов, не зависящей от размеров проекта.

Ключевые слова: формализация, процесс разработки, программное обеспечение, производительность труда, Брукс, мифический человеко-месяц.

Введение

В статье 2000 года [1], посвящённой двадцатипятилетию с момента выхода первого издания книги Фредерика Брукса «Мифический человеко-месяц, или Как создаются программные системы», были отмечены две важные особенности индустрии разработки программного обеспечения:

- «невиданные в истории человечества темпы смены технологий»,
- «лавинообразный рост численности персонала, занятого в отрасли».

Эти две особенности характерны для отрасли и на сегодняшний день.

Брукс в [2] ссылается на исследование [3], выполненное Наусом и Фарром в компании System Development Corporation, результаты которого вполне могут объяснить причины возникновения и даже усиления этих особенностей. Наус и Фарр измерили объём работ как функцию от количества машинных команд на 11 крупных программных проектах. Оказалось, что зависимость имеет вид степенной функции с показателем 1,5. Для наглядности, продемонстрируем результаты этого исследования в виде графика на рисунке 1.

На рисунке 1 ось ординат — объём работ, измеряемый в человеко-месяцах, ось абсцисс — количество машинных команд в тысячах, составляющих программный проект. Сплошная линия — график зависимости объёма работ от количества машинных команд, составляющих программный проект, полученный

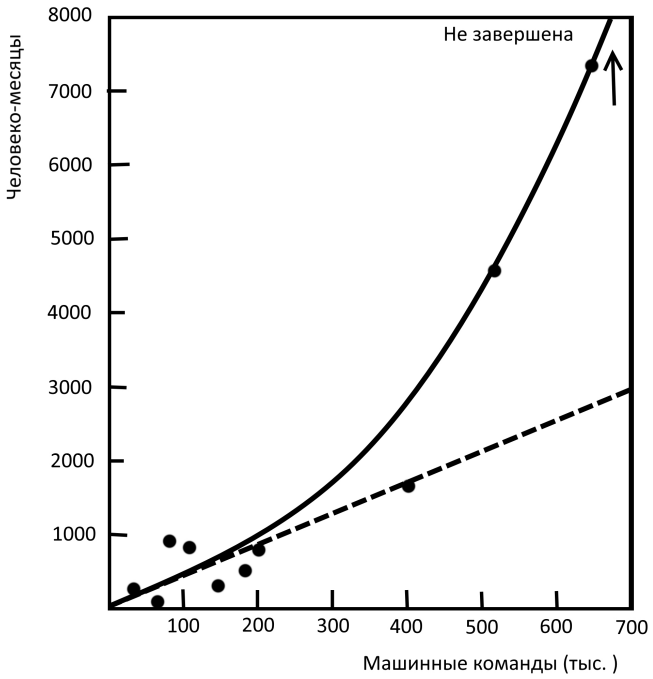

Рис. 1. График зависимости объёма работ от количества машинных команд

Наусом и Фарром, — является экстраполяцией эмпирических данных. Пунктирная линия показывает, как бы выглядел график зависимости объёма работ, если бы производительность труда не зависела об общего размера проекта.

Как видно из графика, производительность труда программистов может снижаться по мере роста размера программы. Опираясь на данное предположение, можно, например, объяснить следующие факты из области разработки программного обеспечения:

- частая смена технологий, выпуск новых версий, переписанных заново — начинать с нуля дешевле и быстрее, чем добавлять новый функционал в существующий продукт;
- дефицит кадров — чтобы сохранять один и тот же темп выпуска нового функционала, по мере роста размера проекта, требуется всё больше сотрудников;
- разбиение проекта на итерации — чем меньше сроки планирования, тем меньше разница между планируемой (постоянной) и фактической производительностью, тем точнее оценка сроков разработки;
- практика измерения velocity, применяемая в Agile-проектах, — это кусочно-линейная аппроксимация степенной функции (сплошная линия на рисунке 1). Velocity — это коэффициент, показывающий разницу между планируемым и фактическим объёмом работ. Используется для улучшения точности оценки предстоящего объёма работ: оценка, сделанная проектной командой, умножается на текущий velocity. Как правило,

velocity пересчитывается в конце каждой итерации;
- за программистами закрепилась устойчивая репутация, что они срывают сроки, в том числе и потому, что прогнозы сроков даются исходя из предположения по постоянной производительности труда (пунктирная линия на рисунке 1).

Измерения, сделанные Наусом и Фарром [3] более чем 50 лет назад, были проведены только для 11 программных проектов, которые, по сегодняшним меркам, нельзя назвать крупными, поэтому нельзя утверждать, что выявленная эмпирическая закономерность справедлива хотя бы для некоторой части современных программных проектов.

Соответственно, возникают вопросы:

- Существует ли эффект падения производительности труда программиста с ростом размера проекта при использовании современных технологий?
- Если да, то можно ли его формально обосновать?
- Можно ли определить условия, при которых производительность труда не будет падать?

В настоящей статье предлагается формализация процесса разработки программного кода как процесса редактирования текста программы и выводится достаточное условие поддержания постоянной производительности труда программиста на всём протяжении проекта.

1. Идея

Прежде чем переходить к строгим математическим рассуждениям, опишем неформально идею.

Программа — это набор слов (который также является словом), записанных на формальном языке. Каждое слово — это последовательность лексем — символов алфавита формального языка. Когда программист вносит изменения в текст программы, то он либо добавляет новое слово, либо удаляет или редактирует существующее, добавляя или удаляя лексему.

Следовательно, можно выделить набор атомарных операций, которые модифицируют текст программы неделимым образом. Определим трудоёмкость разработки каждого слова программы как количество атомарных операций, которые были сделаны для записи этого слова. Трудоёмкость написания текста программы — это суммарная трудоёмкость написания всех его слов, в том числе и тех, которые были удалены в процессе редактирования.

Оценим количество слов программы в зависимости от количества выполненных атомарных операций. Это и будет производительность труда.

2. Атомарные операции

Пусть L — формальный язык над некоторым алфавитом Σ. Множество всех слов над алфавитом Σ будем обозначать как Σ^*, а сами слова символами греческого алфавита α, β, γ, длину слова α – как $|\alpha|$. Напомним, что Σ^* является

полугруппой, то есть

$$\forall \alpha, \beta \in \Sigma^* : \alpha\beta \in \Sigma^*.$$

Пустое слово в Σ^* будем обозначать как ϵ.

Определение 1 (Удаление символа)**.** Будем говорить, что слово α' получено удалением символа $a \in \Sigma$ из слова $\alpha = \beta a \gamma$, где $\beta, \gamma \in \Sigma^*$, причём одно из слов β или γ может быть пустым тогда и только тогда, когда α' можно записать в виде $\alpha' = \beta\gamma$.

Определение 2 (Добавление символа)**.** Будем говорить, что слово α' получено добавлением символа $a \in \Sigma$ в слово $\alpha = \beta\gamma$, где $\beta, \gamma \in \Sigma^*$, причём одно из слов β или γ может быть пустым тогда и только тогда, когда α' можно записать в виде $\alpha' = \beta a \gamma$.

Пример 1. Рассмотрим алфавит языка C++. Слово

```
void f()
{
    int
}
```

получено добавлением символа int из слова

```
void f()
{
}
```

И наоборот, второе слово может быть получено из первого удалением символа int.

Мы хотим ввести формальное определение процесса разработки над языком L. В рамках этого процесса могут появляться одинаковые слова из Σ^*. Чтобы их различать между собой, будем рассматривать не сами слова, а пары (α, n), где $\alpha \in \Sigma^*$, $n \in \mathbb{N}$, и обозначать такие пары $\overline{\alpha}$, а само слово α называть словом пары $\overline{\alpha}$, если будет необходимо.

Здесь и далее по тексту полагаем, что множество натуральных чисел $\mathbb{N}$ содержит 0.

Пусть

$$S = \{\overline{\alpha}_1, \overline{\alpha}_2, \ldots, \overline{\alpha}_n\},$$

— некоторое множество пар (α_i, n_i), где $\alpha_i \in \Sigma^*$, $n_i \in \mathbb{N}$, $i \in \mathbb{N}$, $i \in (1, 2, \ldots, n)$, при этом все n_i попарно различны между собой.

Определим на множестве пар S четыре типа операций:

1. Говорят, что множество пар S' получено добавлением в S пары (α, k), где $\alpha \in \Sigma^*$, $|\alpha| = 1$, $k \in \mathbb{N}$, если и только если

   $$S' = S \cup \{(\alpha, k)\},\ (\alpha, k) \notin S,$$

   при этом $\nexists (\beta, t) \in S : k = t$.

2. Говорят, что множество пар S' получено из S удалением пары (α, k), где $\alpha \in \Sigma^*$, $k \in \mathbb{N}$, если и только если

$$S' = S \setminus \{(\alpha, k)\},\ (\alpha, k) \in S.$$

3. Говорят, что множество пар S' получено из S заменой пары (α, k), где $\alpha \in \Sigma^*$, $k \in \mathbb{N}$, на пару (α', k), где слово α' было получено из слова α добавлением символа $a \in \Sigma$ в смысле определения 3, если и только если

$$S' = S \setminus \{(\alpha, k)\} \cup \{(\alpha', k)\},\ (\alpha, k) \in S.$$

4. Говорят, что множество пар S' получено из S заменой пары (α, k), где $\alpha \in \Sigma^*$, $k \in \mathbb{N}$, на пару (α', k), где слово α' было получено из слова α удалением символа $a \in \Sigma$ в смысле определения 2, если и только если

$$S' = S \setminus \{(\alpha, k)\} \cup \{(\alpha', k)\},\ (\alpha, k) \in S.$$

Замечание 1. $\forall \alpha \in \Sigma^*$ существует такая последовательность операций, результатом применения которых является пара (α, n) для некоторого $n \in \mathbb{N}$.

Доказательство. Пусть $\alpha \in \Sigma^*$. Предположим, что $\alpha = a_1 a_2 \dots a_n$, где $n > 0$. Тогда возьмём следующий набор операций. С помощью операции типа 1 добавим пару (a_1, n). Затем для каждого символа $a_i, i > 1$ с помощью операции типа 3 заменим пару $(a_1 a_2 \dots a_{i-1}, n)$ на пару $(a_1 a_2 \dots a_{i-1} a_i, n)$. ■

Замечание 2 (Корректность набора операций)**.** Пусть S — произвольное множество пар. Тогда существует последовательность операций, результатом применения которых является данное множество пар.

Доказательство. Выполняется индукцией по количеству пар в множестве S и применением предыдущего замечания к каждой паре. ■

3. Процесс разработки

Пусть $\hat{L}$ — подмножество слов языка программирования L над некоторым алфавитом Σ. Подмножество $\hat{L}$ символизирует те ограничения, которые накладывают программисты на используемый язык в связи с выбранной методологией программирования, например, процедурной, объектно-ориентированной и т. д. или стандартами кодирования, принятыми в рамках процесса разработки, например, запрет на использование глобальных переменных.

Определение 3. Пусть

$$S = \{\overline{\alpha}_i | i \in (1, 2, \dots, n)\},$$

где $n \in \mathbb{N}$, $\alpha_i \in \Sigma^*$ — некоторое множество пар. S называется программой на языке L тогда и только тогда, когда $\forall \overline{\alpha}_i : \alpha_i \in \hat{L}$.

Определение 4. Последовательность множеств пар:

$$P_0 \xrightarrow{c_1} P_1 \xrightarrow{c_2} P_2 \xrightarrow{c_1} \ldots,$$

где $P_0 = \varnothing$, P_i, где $i > 0$ — множество пар, P_i получено из P_{i-1}, с помощью операции c_i одного из перечисленных выше типов, при этом для всех операций типа 1 все добавленные пары имеют вид (α, k), где $\alpha \in \Sigma^*$, k — номер шага, на котором операция была выполнена, называется процессом разработки Pr.

Определение 5. Говорят, что программа P получена с помощью процесса разработки Pr, если $\exists i > 0 : P_i = P$.

Все дальнейшие рассуждения будем проводить в предположении, что имеется некоторый процесс разработки Pr:

$$P_0 \xrightarrow{c_1} P_1 \xrightarrow{c_2} P_2 \xrightarrow{c_1} \ldots.$$

Замечание 3. В силу определения процесса разработки для любого $k \in \mathbb{N}$ либо не существует пары (α, k), которая была создана в процессе разработки, либо существует и при том только одна.

Определим для удобства последовательность множеств удалённых пар как:

1. $D_0 = \varnothing$,
2. $D_i = D_{i-1} \cup \{\overline{\alpha}\}$, если $\overline{\alpha}$ было удалено на шаге i из множества P_{i-1} с помощью операции типа 2, и $D_i = D_{i-1}$ в противном случае.

Предложение 1. $\forall n \in \mathbb{N} : P_n \cap D_n = \varnothing$.

Доказательство проводится по индукции в силу построения множества пар P_n.

4. Скорость процесса разработки

Далее определим на множестве $\Sigma^* \times \mathbb{N}$ — декартовом произведении множеств Σ^* и $\mathbb{N}$ — последовательность функций

$$F_i : (\Sigma^* \times \mathbb{N}) \to \mathbb{N}, i \in 1, 2, ..., n :$$

1. $F_0(\Sigma^* \times \mathbb{N}) = 0$.
2. Предположим, что все функции F_j, $j \leqslant i$ определены. Пусть $\overline{\alpha} \in \Sigma^* \times \mathbb{N}$ — некоторая пара. Тогда
   - $F_i(\overline{\alpha}) = 1$, если пара $\overline{\alpha} = (\alpha, i)$ была добавлена в P_i с помощью операции типа 1 на шаге i.
   - $F_i(\overline{\alpha}) = F_{i-1}(\overline{\alpha}')+1$, если пара $\overline{\alpha} = (\alpha, k)$ была получена из $\overline{\alpha}' = (\alpha', k)$ с помощью операции типа 3 или типа 4 на шаге i.

- $F_i(\overline{\alpha}) = F_{i-1}(\overline{\alpha}) + 1$, если пара $\overline{\alpha} = (\alpha, k)$ была удалена на шаге i из набора P_{i-1} с помощью операции типа 2.
- $F_i(\overline{\alpha}) = F_{i-1}(\overline{\alpha})$, в противном случае.

Предложение 2.

$$\forall \overline{\alpha} \in \Sigma^* \times \mathbb{N}\, \forall k, m \in \mathbb{N} : k > m \Rightarrow F_k(\overline{\alpha}) \geqslant F_m(\overline{\alpha}).$$

Следует из построения набора функций $F_i : \Sigma^* \times \mathbb{N} \to \mathbb{N}$, $i \in \mathbb{N}$.

Предложение 3. *Пусть* $\overline{\alpha} \in \Sigma^* \times \mathbb{N}$. *Тогда последовательность* $\{F_i(\overline{\alpha}), i \in \mathbb{N}\}$ *либо сходится к некоторому* $C \in \mathbb{N}$, *либо* $\forall C \in \mathbb{N}\, \exists k > 0 : F_k(\overline{\alpha}) > C$.

Следует из определения функций F_i, $i \in \mathbb{N}$ и предложения 2.

Определение 6. Пусть $\overline{\alpha} \in \Sigma^* \times \mathbb{N}$. Тогда асимптотической трудоёмкостью написания слова α в процессе разработки Pr называется $\lim\limits_{i \to \infty} F_i(\overline{\alpha})$.

Определение 7. Определим $F_{Pr} : \Sigma^* \times \mathbb{N} \to \mathbb{N}$ — асимптотическую трудоёмкость процесса разработки Pr как $F_{Pr}(\overline{\alpha}) = \lim\limits_{i \to \infty} F_i(\overline{\alpha})$.

Определение 8. Пусть $F : \mathbb{N} \to \mathbb{R}$ — некоторая функция из множества $\mathbb{N}$ в $\mathbb{R}$. Говорят, что процесс разработки Pr обладает скоростью F тогда и только тогда, когда

$$\lim_{n \to \infty} \frac{|P_n|}{F(n)} = C,$$

где $C > 0$.

5. Достаточное условие линейной скорости процесса разработки

Пусть задан некоторый процесс разработки Pr

$$P_0 \xrightarrow{c_1} P_1 \xrightarrow{c_2} P_2 \xrightarrow{c_1} \dots$$

В терминах предыдущего параграфа

Предложение 4.

$$\forall n \in \mathbb{N}\, |P_n| + |D_n| \leqslant n.$$

Доказательство утверждения выполняется индукцией по номеру n.

Предложение 5.

$$\forall n \in \mathbb{N} \sum_{\overline{\alpha} \in P_n \cup D_n} F_n(\overline{\alpha}) = n.$$

Доказательство утверждения выполняется индукцией по номеру n.

Предложение 6. *Предположим, что* $\exists C > 0, C \in \mathbb{N}\ \forall n \in \mathbb{N}\ \forall \overline{\alpha} \in P_n \cup D_n : F_n(\overline{\alpha}) < C$. *Тогда*

$$\frac{n}{C} \leqslant |P_n \cup D_n| \leqslant n.$$

Доказательство. Ограничение сверху следует из предложений 1 и 4. Ограничение снизу прямо следует из условия и предложения 5. ■

Предложение 7. *Предположим, что* $\exists C > 0,\ C \in \mathbb{N}\ \forall n \in \mathbb{N}\ \forall \overline{\alpha} \in P_n \cup D_n : F_n(\overline{\alpha}) < C$. *Тогда*

$$|P_n \cup D_n| \sim O(n).$$

Доказательство. Следует из предложения 6 и свойств предела числовой последовательности. ■

Теорема 1 (Достаточное условие линейной скорости процесса разработки)**.** *В условиях предложения 7 предположим, что* $\lim\limits_{n\to\infty} \frac{|D_n|}{|P_n|} = C$, *где* $C \geqslant 0$. *Тогда*

$$|P_n| \sim O(n).$$

Доказательство. Следует из предложений 1, 7 и свойств предела числовой последовательности. ■

Рассмотрим теперь случай, когда асимптотическая трудоёмкость разработки каждого слова ограничена некоторой константой, но при этом не существует такой константы, которая ограничивает сразу асимптотическую трудоёмкость разработки всех слов. Представим данное условие в виде двух высказываний:

$$\forall n \in \mathbb{N}\ \forall \overline{\alpha} \in P_n \cup D_n\ \exists C > 0 : F_n(\overline{\alpha}) < C. \tag{1}$$

$$\forall C > 0\ \exists k \in \mathbb{N}\ \exists \overline{\beta} \in P_k \cup D_k : F_k(\overline{\beta}) > C. \tag{2}$$

Пусть $C > 0$ — некоторая константа. Построим множество S пар $\overline{\alpha}$ следующим образом:

1. В силу предположения (2) $\exists k \in \mathbb{N}\ \exists \overline{\beta} \in P_k \cup D_k : F_k(\overline{\beta}) > C$. Тогда определим $S_0 = \{\overline{\beta}\}$.
2. Предположим, что для всех $j < i$ множества S_j уже построены. Среди всех $\overline{\beta} : \exists k \in \mathbb{N}\ \exists \overline{\beta} \in P_k \cup D_k : F_k(\overline{\beta}) > C + i$ выберем, что $\forall j < i : \overline{\beta} \notin S_j$. Если такое $\overline{\beta}$ существует (доказательство существования см. в предложении 8 ниже по тексту), то определим $S_i = S_{i-1} \cup \{\overline{\beta}\}$.

Замечание 4.

$$\forall i \in \mathbb{N} : |S_i| = i.$$

Замечание 5. $\forall i > 0 : S_{i-1} \subset S_i$.

Предложение 8. *В условиях пункта 2 построения множества* S $\overline{\beta}$ *существует.*

Доказательство. От противного. Предположим, что $\overline{\beta}$ не существует, тогда в силу (2), замечания 5 и сделанного предположения

$$\forall\overline{\gamma}\ \exists k \in \mathbb{N} : (\overline{\gamma} \in P_k \cup D_k) \wedge (F_k(\gamma) > C + i) \to \overline{\gamma} \in S_{i-1},$$

где $\to$ — это импликация. Поскольку множество S_{i-1} — конечное, и для каждой пары $\overline{\alpha}$ из S_{i-1} в силу (1) существует константа C_α, что $F_n(\overline{\alpha}) < C_\alpha$. Тогда мы можем определить общую константу $C_g = \max\limits_{\overline{\alpha} \in S_{i-1}} C_{\overline{\alpha}}$, что $\forall n \in \mathbb{N}\ \forall\overline{\alpha} \in P_n \cup D_n$: $F_n(\overline{\alpha}) < C_g$. Получаем противоречие с (2), следовательно, $\overline{\beta}$ существует. ■

Определим множество S как

$$S = \bigcup_{i \in \mathbb{N}} S_i.$$

Предложение 9. *Множество S бесконечное.*

Доказательство. Следует из построения множества S в силу замечания 5. ■

Теорема 2. *Пусть (1) и (2) справедливы для процесса разработки. Тогда*

$$\forall C > 0\ \exists k \in \mathbb{N} : |P_k \cup D_k| > C.$$

Доказательство. Пусть $C > 0$ — некоторая константа, S — множество, построенное, как описано выше. В силу предложения 9 $\exists k \in \mathbb{N} : |S_k| > C$. В силу построения множества S : $\exists m \in \mathbb{N} : S_k \subset P_m \cup D_m$. Следовательно, $|P_m \cup D_m| > C$. ■

Условий (1) и (2) недостаточно, чтобы процесс разработки обладал линейной скоростью. Рассмотрим следующий пример.

Пусть C — некоторая константа. Рассмотрим процесс разработки Pr_{ex}, при котором последовательно был получен следующий набор слов:

$\overline{\alpha}_1 : F_{Pr}(\overline{\alpha}_1) = C$, $\overline{\alpha}_2 : F_{Pr}(\overline{\alpha}_2) = C{+}1$, $\overline{\alpha}_3 : F_{Pr}(\overline{\alpha}_3) = C{+}2, \ldots$ Следовательно,

$$\sum_{i=1}^{k} F_k(\overline{\alpha}_i) = Ck + \frac{k(k-1)}{2}. \tag{3}$$

Тогда набор пар

$$(\alpha_1, 1), (\alpha_2, C+1), (\alpha_3, C+2), \ldots, \left(\alpha_k, C + \frac{k(k-1)}{2}\right)$$

составляет множество пар P_n для некоторого шага n процесса Pr. Заметим, что $|P_n| = k$. В силу предложения 5 и формулы (3)

$$Ck + \frac{k(k-1)}{2} = n.$$

Нетрудно показать, что в таком случае имеет место

$$|P_n| \sim O(\sqrt{n}).$$

Другими словами, рассмотренный нами в примере процесс разработки Pr_{ex} имеет скорость $\sqrt{n}$.

Предложение 10. *Процесс разработки Pr_{ex} существует.*

Доказательство. Рассмотрим функцию

```
void f1()
{
}
```

Данная функция состоит из 7 лексем и согласно нашему определению трудоёмкости при условии, что мы только последовательно добавляли лексемы, её трудоёмкость составляет 6+1.

Пусть $k \in N, k > 1$. Рассмотрим функцию

```
void f1...1()
{
  ;;...;
}
```

Имя этой функции состоит из f и k единиц, а тело функции содержит k пустых операторов. То есть данная функция состоит из $6 + k$ лексем и, следовательно, при условии, что мы последовательно добавили k лексем, трудоёмкость её разработки составит $6 + k$. Мы показали, как построить функцию для любого заданного k. ■

6. Полученные результаты и практика

Достаточное условие линейной скорости процесса разработки (Теорема 1) подтверждается практиками разработки программного обеспечения:

- Мартин Фаулер в [4] с. 54–56 даёт рекомендацию: «классы, процедуры, функции должны быть небольшими».
- The Open-Closed Principle [5]: «Программные сущности (классы, модули, функции и т.п.) должны быть закрыты от изменений...»

Важность ограничений на модификацию программного кода легко проиллюстрировать на следующем примере. Рассмотрим рисунок 2. Прямоугольники обозначают процедуры, а стрелки — какие-либо зависимости одной процедуры от другой. Например, если одна процедура содержит в своём теле вызов второй, то первая процедура зависит от второй. Основание стрелки — это процедура, которая зависит от второй, на которую показывает стрелка. Предположим, что нам необходимо внести изменения в код процедуры, обозначенной цифрой 1. Кроме тестирования самой процедуры 1, нам придётся, как минимум, протестировать, а, возможно и вносить правки в код процедур 2 и 3. В случае правок в процедуре 2, придётся проверять, и, возможно изменять процедуру 4, а в случае правок в процедуре 3 — выполнить те же действия для процедур 5 и 6. И так далее.

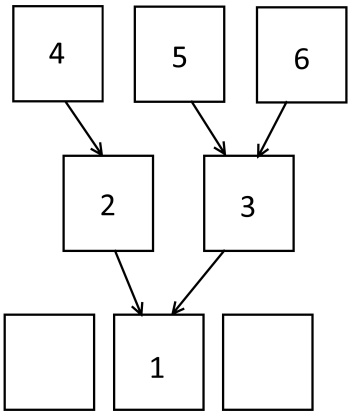

Рис. 2. Расходящиеся модификации

Этот пример показывает, что при внесении изменений в какой-либо код могут понадобиться правки в коде, который использует модифицируемый. При этом объём кода, работоспособность которого может быть нарушена, не зависит и никак не может быть ограничен разработчиком кода, для которого потребовалась модификация.

Существует немало примеров, когда небольшие правки затрагивали большое число программных проектов по всему миру:

- Проблема 2000 года. Проблема была связана с форматом хранения даты, который был рассчитан на хранение диапазона дат с 1970 по 1999 годы. Никто не мог сказать, как наступление 1 января 2000 года отразится на работоспособности программного обеспечения. Серьёзных последствий удалось избежать лишь потому, что эта проблема была поднята за несколько лет до наступления 2000 года: и производители оборудования, а затем и программисты с учётом темпов смены технологий заблаговременно внесли изменения в программный код.
- Февраль 2016 года [7].
  Разработчик функции, которая отрезает пробелы слева, удалил соответствующий npm пакет из репозитория javascipt пакетов. В результате существенное число популярных javascript проектов перестало собираться. Многие разработчики даже не догадывались, что их проекты зависят от этой функции, так как зависимость была у сторонних проектов, включённых в проект.
- Страуструп, создатель языка C++, в [6] признается, что некоторые полезные и изящные конструкции не вошли в состав стандарта языка C++, так как они могли нарушить обратную совместимость.

Заключение

В настоящей статье было определено достаточное условие линейной скорости процесса разработки программного обеспечения, которое не зависит от языка и используемой парадигмы программирования, и которое, в основном, сводится к требованию, чтобы на разработку каждой программной сущности: класса, процедуры, функции и т. д. – не должно уходить более, чем заранее фиксированное число операций.

Кроме этого, показано, что требование, чтобы каждая программная сущность лишь не модифицировалась, то есть имела конечную трудоёмкость разработки, недостаточно, чтобы процесс разработки обладал линейной скоростью.

Литература

1. Колесов А. Мифический человеко-месяц: двадцать пять лет спустя // PCWeek. 2000. № 7(229). URL: https://www.pcweek.ru/themes/detail.php?ID=53642.
2. Чапел Х., Брукс Ф. Мифический человеко-месяц, или Как создаются программные системы. СПб. : Символ-Плюс, 2010.
3. Nanus B., Farr L. Some cost contributors to large-scale programs // AFIPS Proc. SJCC. Spring 1964. Vol. 25. P. 239–248.
4. Фаулер М. Рефакторинг: улучшение существующего кода. СПб. : Символ-Плюс, 2003.
5. Martin R. The Open-Closed Principle // Object Mentor. 1996. URL: http://www.objectmentor.com/resources/articles/ocp.pdf.
6. Страуструп Б. Дизайн и эволюция языка C++. СПб. : ДМК Пресс, 2006.
7. Williams C. How one developer just broke Node, Babel and thousands of projects in 11 lines of JavaScript. URL: https://www.theregister.co.uk/2016/03/23/npm_left_pad_chaos/.

ABOUT THE FORMALIZATION OF THE SOFTWARE DEVELOPMENT PROCESS

E.A. Tyumentcev
CEO, e-mail: etyumentcev@hwdtech.ru

LLC Hello World! Technologies

Abstract. In the book "Mythical man-month, or How software systems are created," Brooks refers to several studies on the basis of which it can be concluded that the productivity of the work of programmers, measured in the number of lines of code per time, decreases as the size of the program project grows. In this paper, we describe the formalization of the software development process, as the process of editing the source code of the program, with the help of which the laboriousness function is defined, and a sufficient condition for the constant productivity of the programmers, independent of the size of the project, is derived.

Keywords: formalization, development process, software, productivity, Brooks, mythical man-month.

Дата поступления в редакцию: 25.04.2017