Заметки о „грамотном программировании” в текстовом процессоре LyX

Дьяков С.Е.

26 сентября 2012 г.

Аннотация

Описан опыт использования текстового процессора LyX как среды создания „грамотных” программ с использованием пакета NoWeb.

Содержание

1 Введение

Программный продукт, в общем случае, включает с себя не только тексты программ, но и различную документацию: проектную, техническую, пользовательскую и маркетинговую. В зависимости от объема программы, требований к программе и выразительности технических средств [?], некоторые виды данной документации могут отсутствовать, или включаться в исходный текст программы в виде комментариев.

Проектная и техническая документация включает информацию о оказывающих влияние на программу в целом архитектурных решениях, принимаемых на этапах планирования, проектирования и разработки. Выделение документации в отдельные документы, естественное и неизбежное в случае больших проектов, приводит к серьезной проблеме синхронизации документации и исходного кода. Значимость данной проблемы в плане технической документации и один из методов решения — создание самодокументируемых программ — описана Фредериком Бруксом [?].

Одним из способов „обхода" проблемы синхронизации проектной документации и текста программы, является использование „грамотного программирования" — методики которая предлагает программисту объединить проектную и техническую документацию вместе и исходным текстом программы в рамках одного единого документа.

В рамках данной статьи будет кратко описана идеология и история грамотного программирования, приведен пример программирования в текстовом процессоре LyX.

1.1 Краткий обзор грамотного программирования

„Грамотное программирование" (в дальнейшем ГП) — это методика написания программ делающая упор не на написании собственно кода программы, а на *последовательном письменном* рассмотрении всех тех задач которые решает программист получивший задание, от начала и до самого конца. При этом текст собственно программы генерируется автоматически на основе полученного документа.

ГП может использоваться (и используется) для написания программ в которых объем и/или ценность знаний относительно велика по сравнению с объемом кода. При этом время написания программ увеличивается, но возрастает и „время жизни" программы, снижается стоимость сопровождения.

Первыми системами грамотного программирования были системы WEB [?]и CWEB [?], которые использовались для создания системы верстки TEX и шрифтовой системы METAFONT. Данный опыт, был, по мнению автора, удачным так-как указанные программы, несмотря на их сложность, были созданы силами одного программиста за короткое время и получили широкое распространение. Тексты данных программ были опубликованы в виде отдельных книг [?, ?]. В интервью, говоря о влиянии методологии ГП на создание этих программ и других программ, Д. Кнут говорит:

> Этот подход не только позволил мне писать и поддерживать программы быстрее и надежнее, чем когда бы то ни было раньше, и он не только был для меня самым большим источником удовольствия, начиная с 1980-х гг. – он иногда оказывался незаменимым. Некоторые из моих основных программ, такие как метасимулятор MMIX, не могли бы быть написаны с применением любой другой методологии, о которой я когда-либо слышал. Сложность была просто чересчур устрашающей, чтобы с ней можно было справиться на основе моих ограниченных умственных возможностей; без применения грамотного программирования все предприятие потерпело бы полную неудачу.

Для того, чтобы использовать ГП необходимы средства извлечения текста программ из документа, и дополнительной подготовки документа к печати. Совокупность этих средств я называю системой грамотного программирования.

Особенностями первых систем грамотного программирования были [?]:

1. использование языка разметки TEX, как средства оформления документации;

2. широкое использование макросов как средств обобщенного программирования;

3. автоматическое создание указателей переменных, функций, типов и модулей для того, чтобы облегчить анализ созданных программ.

Опыт использования грамотного программирования с точки зрения повторного использования программ и обучения был описан С.Бартом и его последователями. Например, в статьях [?, ?] описан опыт переноса системы TEX и родственных программ (MetaPOST, MetaFONT) на новую платформу — систему Data General AOS, при этом упоминается, что полный перенос сложной и большой программы на новую аппаратную платформу занял три дня.

Опыт использования грамотного программирования при обучении программированию и решению задач (CS1) также оказался положительным [?], при этом была обнаружена зависимость между умением использовать грамотное программирование и пониманием следующего курса — CS2.

Другой эксперимент, связанный с разработкой и поддержкой разработки программ, был проведен и описан Норманом Рамси [?]. В рамках эксперимента, в ходе которого небольшая группа программистов разрабатывала систему семантического анализа программ на языке Ада, была подтверждена эффективность методики грамотного программирования, которая облегчила взаимодействие между программистами, способствовала созданию высококачественной и полезной документации. При этом участники заявляли о следующих недостатках технических средств ГП: сложность системы TEX, как системы подготовки документации; неудобство текстового редактора emacs; отсутствие практической пользы от средств художественного оформления текстов программ, которые, к тому-же игнорируют отступы и разрывы строк сделанные программистами, исходя из логики программы.

Технически грамотное программирование состоит из нескольких независимых частей, каждая из которых может отсутствовать. К этим частям, по крайнем мере, относятся: возможность оформления комментариев, использования формул, таблиц, изображений и т.п. в комментариях, создание текста программы и технической документации к нему из одного единого документа, обобщенного программирования с помощью системы макросов, автоматическое типографское оформление текста программ и функций, автоматическое создание списка функций, переменных и модулей, автоматический контроль ссылок на переменные и функции, использование системы верстки TEX/LATEX для оформления документации.

Так-как требования и вкусы программистов различаются, а создание системы грамотного программирования сравнительно несложная задача, было разработано большое количество систем грамотного программирования.

В настоящее время эксплуатируются следующие системы с полной реализацией всех частей грамотного программирования для одного или нескольких языков программирования: WEB[?], CWEB [?], FWEB [?].

Между тем, распространение получили системы грамотного программирования без возможности вывода кода программ с выделением синтаксиса и некоторые возможности WEB для индексации, но с поддержкой макросов: FunnelWEB [?], nuweb [?], fangle [?] и некоторые другие.

Норман Рамси, исходя из результатов своего эксперимента [?], предпочел отказаться как от макросов, так и от оформления исходного кода выводимых программ и создал свою систему грамотного программирования noweb [?]. Данная система отличается простотой и внутренней ортогональностью. Возможно, это самая простая система грамотного программирования из возможных, для её описания достаточно двух-трех абзацев.

Помимо систем грамотного программирования общего назначения, существуют, используются системы специальные, ориентированные не только на один язык программирования, но и на одну узкую область использования, например для статистической обработки данных — система Sweave и [?].

Наконец, поддержка некоторых элементов грамотного программирования встраивается как с среды разработки, например LEP[?] для IDE Eclipse, так и в языки программирования, такие как Haskel [?].

Для эффективного создания программ недостаточно иметь систему грамотного программирования — нужна среда для создания программ. В качестве среды могут использоваться как

обычный текстовой редактор, возможно настроенный соответствующим образом как emacs [?] и vim, так и специализированные редакторы, ориентированные именного на литературное программирование, такие как Leo [?, ?], RWINED [?] и HOPS [?], предлагающие расширенные средства навигации по документу и визуализации диаграмм, графиков и т.п.

Эффективное создание программ предусматривает полную концентрацию программиста, на том, что он делает в данный момент, и устранение всего, что мешает концентрации [?]. Для создания программ в *некоторых областях*, эффективным может быть использование обычного текстового редактора. Возможно, старейшей их таких система является система WinWordWeb [?], которая в настоящее время не поддерживается, или, например, использование простой системы грамотного программирования noweb, вместе с текстовым процессором LyX [?].

1.2 Отказ от попыток понравиться всем и захватить мир

Подход грамотного программирования используется сравнительно редко [?] и считается экзотическим. Говорят, что для того, чтобы писать грамотные программы, надо уметь не только программировать, но и писать ясные, структурированные тексты [?], что два этих умения редко встречаются у одного и того-же программиста.

С этим можно поспорить.

Грамотное программирования, возможно, действительно не подходит для программирования массового. С другой стороны, не существует стиля или методики программирования, будь то экстремальное программирования, или методы быстрой разработки или разработки через тестирование который подходил бы для всех задач и для всех программистов.

Данный метод идеально работает в тех случаях, когда надо рассмотреть и задачу, и алгоритм и решение, когда объяснение само по себе является ценностью.

Он хорош для использования в научной среде [?], в тех случаях когда решение надо обосновывать.

Существуют примеры успешного использования данного подхода для создания книг по программированию или описанию тех или иных методик расчетов [?, ?, ?, ?].

Поэтому хотя грамотное программирование, конечно, никогда не станет превалирующим методом программирования, но никогда и не умрет. Оно постепенно найдет свою нишу, и сделает жизнь там немного лучше.

1.3 Опыт использования ГП в научных исследованиях, прикладном программировании и обучении программированию. Будущее ГП.

1.4 Узкая постановка задачи. Цели исследования

Thinking with Style (Daniel Lanovaz, Los GatosТехнология грамотного программирования (далее в тексте — ГП) описана в книге Д. Кнута „Literate programming”. На русском языке краткое описание ГП опубликовал Андрей Зубнинский [?].

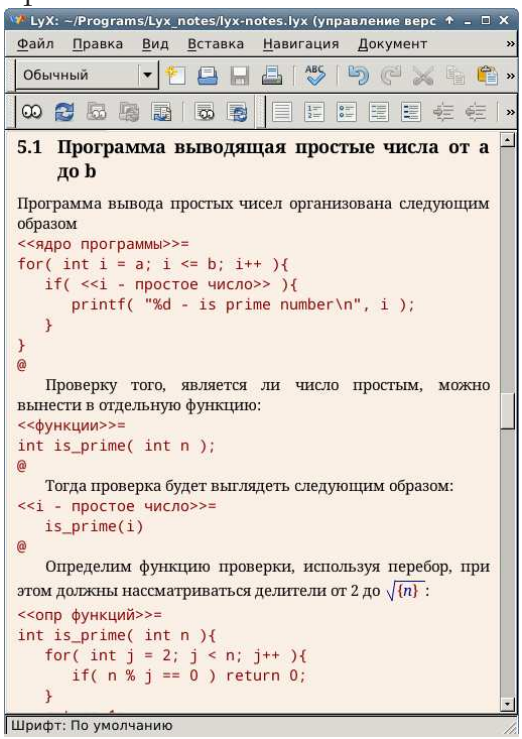
Инструментальные средства ГП были предложены в конце 80-х годов и соответствовали уровню языков и сред того времени. Предполагалось, что документ ГП это текст в формате TeX созданный в обычном текстовом редакторе, при этом в документ включаются кусочки программ и управляющие конструкции системы ГП. Пригодный к печати текст документации и исходный текст программы будут генерироваться подготовленного документа. Язык ГП был достаточно развитым, он позволял получать текст программы полиграфического качества для распространенных языков программирования, поддерживал использование макросов.

Опыт использования ГП для разработки значительной по объему программы (33 000 строк) силами группы из семи сотрудников описан в статье Нормана Рамсея [?]. Данный опыт показал как перспективность данной технологии, так и недостатки её использования и реализации. В частности, ориентация на высокое качество печатного варианта программы не оправдывает себя, и вредит в том случае, если программист не может фактически управлять размером отступов и переносами строк текста программы. Далее, использование сложного языка верстки TeX мешает использованию технологии, к тому-же TeX недостаточно выразителен и не содержит средств для вставки диаграмм, графиков и т.п. С другой стороны, было обнаружено что качество и кода и

документации растут, если программист описывает что, зачем и как делает та или иная часть программы. В конечном итоге, к качеству и полезности документации к программе не было никаких претензий.

В настоящее в время, необходимость изучения дополнительного (и сложного) языка верстки, мешает использованию „грамотного программирования” даже областях, для которых оно идеально подходит, а сложность и непривычность инструментальных средств делает его применение достаточно редким, о чем и говорится в статье А. Зубнинского. Необходимо сказать, что проектирование удобных инструментальных средств для грамотного программирования чрезвычайно сложная задача, а их наличие — жизненно важно для использования технологии.

Рис. 1: Пример создания программы в LyX



Поэтому продолжаются попытки создания инструментальных средств ГП, к которым можно отнести текстовой редактор Leo [?, ?], расширения редакторов emacs [?] и vim, редакторов RWINED [?] и HOPS [?], средства ГП встроены в язык программирования Haskell [?], существует версия языка программирования статистических расчетов R — Sweave [?], ориентированная именно на ГП. Все это говорит о том, что ГП действительно позволяет сконцентрироваться на болевой точке современного программирования и повысить качество программ, хотя и ценой сравнительно больших первоначальных затрат.

Использование ГП в образовании описано в статье [?] .

В данной статье описывается опыт использования текстового процессора LyX для ГП [?].

Свободный текстовой процессор LyX , созданный Маттиасом Эттрихом и другими разработчиками, придерживается идеологии WYSIWYM (what you see is what you mean, примерный перевод — „видишь то, что подразумеваешь”), слегка напоминает популярный редактор Microsoft Word, и генерирует файлы для системы TEX [?]. Более того, данный текстовой процессор содержит средства интеграции с простой системой ГП NoWeb [?]. (Несколько более мощная система ГП fangle тоже может использоваться для ГП. Кроме этого LyX содержит средства интеграции и с системой ГП на языке R Sweave.)

Использование текстового процессора LyX для создания программ в стиле „грамотного программирования” целесообразно в некоторых предметных областях, связанных с разработкой методов и алгоритмов обработки данных, что подтверждается практикой авторов. Было обнаружено, что использование текстового процессора удобно при создании ... программ (Рис.1).

В данной статье описан пример использования программы LyX для создания учебных программ.

Эта работа не является в полной мере пионерской. Авторам удалось найти текст лекции Х.Рамачадрана, посвященной использованию LyX для программирования в scilab [?]. Данная работа отличается ориентацией на использование LyX в реальном программировании, что требует создания средств локализации ошибок, интеграции системы сборки программ и т.п.

2 Кратко о текстовом процессоре LyX

LyX — свободный процессор документов, основанный на системе компьютерной вёрстки LATEX, первые версии которого были созданы Маттиасом Эттихом, ведущим разработчиком KDE.

К особенностям программы можно отнести:

1. удобный интерфейс пользователя, напоминающий популярный текстовой редактор Microsoft Word;

2. ориентацию на определение структуры документа, и автоматическую верстку представления на основе механизма шаблонов;

3. наличие средств для использования БД библиографических источников, создания перекрестных ссылок, автоматического формирования предметных указателей, оглавлений и других навигационных элементов;

4. возможность автоматического экспорта текста документа в форматы html, pdf и другие;

5. наличие встроенного самоучителя, переведенного на русский язык;

6. отсутствие возможности создания собственных стилей, так-как набор используемых стилей жестко задан шаблоном документа.

В настоящее время тестовой процессор LyX это мощное и удобное средство создания значительных по объему документов, гарантирующее профессиональное качество печатного документа.

3 Кратко о системе грамотного программирования NoWeb

По сравнению с другими системами грамотного программирования система NoWeb отличается простотой. Исходный файл документа системы — это обычный файл в формате LaTeX и с расширением .nw. В этом файле находятся определения кусочков кода программы — так называемых чанков.

Чанк определяется следующим образом:

```
<<имя чанка>>=
текст чанка строка 1
текст чанка строка 2
    <<имя чанка 2>>
текст чанка строка 2
```

Начинается чанк с новой строки и символов „<<”. Имя чанка может включать пробелы и любые другие символы, доступные в тексте LaTeX. текст чанка завершается строкой с символом „@”. Чанк может включать ссылки на другие чанки, в тексте это `<<имя чанка 2>>`.

Команда `notangle` позволяет выделить из документа чанки с определенным корнем и сохранить их в файл.

Пример документа noweb и полученного текста программы приведен в таблице 1.

Преведенный пример показывает исключительную лаконичность языка NoWeb. Для извлечения текста программы достаточно знать имя корневого чанка (в данном случае это `NOD.cpp`. Особенностью системы является то, что чанк `NOD` определяется несколько раз, при этом результат „раскрутки” чанка включает все его определения, объединенные последовательно. Еще одной особенностью является сохранение ведущих пробелов строк, что позволяется использовать систему NoWeb для создания программ на языке `python`.

4 Установка системы настройка её для создания программ в стиле „грамотного программирования” и начало работы

4.1 Установка и настройка системы грамотного программирования noweb в Ubuntu Linux

```
sudo apt-get install noweb
```

документ NoWeb	команда	текст программы

Программа должна находить
наибольший общий делитель двух
целых чисел с помощью
алгоритма Евклида.
 	
<<NOD.cpp>>=
#include <stdio.h>
 	
<<функции>>
 	
int main(){
  <<основная функция>>
  return 0;
}
@
 	
Получим два числа, НОД которых
необходимо найти.
<<основная функция>>=
int a, b;
scanf("%d %d", &a, &b);
@
 	
Алгоритм вычисления НОД реализуем
с помощью рекурсивной функции.
<<функции>>=
int NOD(int a, int b){
  <<NOD>>
}
@
вызовем её и выведем ответ:
<<основная функция>>=
printf("NOD(%d,%d) = %d\n",
  a, b, NOD(a,b));
@
Заполним определение функции NOD
реализовав рекурсивный алгоритм [1].
1. НОД(0, n) = n;
<<NOD>>=
if(a == 0) return b;
@
2. НОД(m, 0) = m;
<<NOD>>=
if(b == 0) return a;
@
3. НОД(m, m) = m;
<<NOD>>=
if(a == b) return a;
@
4. n>m -> НОД(n, m) = НОД(n % m, m)
<<NOD>>=
if(a > b) return NOD(a%b, b);
@
4. n<m -> НОД(n, m) = НОД(n , m % n)
<<NOD>>=
if(a < b) return NOD(a, b%a);
@
	notangle \
	  -RNOD.cpp \
	  a.nw \
	  >NOD.cpp
		#include <stdio.h>
		
		int NOD(int a, int b){
		  if(a == 0) return b;
		  if(b == 0) return a;
		  if(a == b) return a;
		  if(a > b) return NOD(a%b, b);
		  if(a < b) return NOD(a, b%a);
		}
		
		int main(){
		  int a, b;
		  scanf("%d %d", &a, &b);
		  printf("NOD(%d,%d) = %d\n",
		  a, b, NOD(a,b));
		  return 0;
		}

Таблица 1: Документ NoWeb, и полученная программа

4.2 Установка и настройка системы верстки TEX/Latex и сопутствующих программ в Ubuntu Linux

```
sudo apt-get install texlive-full cm-super lmodern latex2html gv source-highlight tidy
```

4.3 Установка и настройка текстового процессора LYX в Ubuntu Linux

```
sudo apt-get install lyx elyxer ttf-lyx enchant aspell-ru aspell-en aspell
```

4.3.1 Настройка среды в ОС GNU/Linux

Меню → Инструменты → Переконфигурировать

Настройка проверки правописания Меню → Инструменты → Настройки...
Выбрать Настройки языка → Проверка правописания
spellchecker engine: Enchant
другие языки: ru
Выбрать Настройки языка → Язык
Язык пользовательского интерфейса: По умолчанию

Настройка документа Меню → Документ → Настройки...

Класс документа Выбрать класс документа article (Noweb).
Поле „Custom” оставить пустым

Шрифты Для всех начертаний поставить по умолчанию.
Выбрать размер шрифта (11pt).

Язык

Язык: выбрать „Русский”

Кодировка: указать „другой”, выбрать Юникод (utf8)

Кавычки: указать стиль кавычек отличный от „лапок”

Поля Указать размеры полей

Уменьшим размер текста программ В „Преамбула Latex” добавить

```
\@ifundefined{lstset}{}{
\noweboptions{longchunks,smallcode}
\lstset{
basicstyle=\footnotesize,
breaklines=true,
breakatwhitespace=true,
resetmargins=true,
xleftmargin=3em
}
}
\noweboptions{smallcode}
```

Свойства PDF Поставить галочку в поле „Использовать поддержку PDF”.
Во вкладке „Гиперссылки” поставить галочку в поле „разрывать ссылки через строки”.

Смягчим требования к качеству переносов В „Преамбула Latex” добавить

```
\sloppy
```

Настроим связь между LyX-текстом и графическим образом документа dvi Связь между текстовым редактором LyX и просмотрщиком xdvi значительно облегчает исправление ошибок и контроль за внешним видом документа.

Для отображения файлов dvi надо использовать программу xdvi. Для этого надо в Меню → Инструменты → Настройки...→ Обработка файлов → File formats выбрать формат dvi, ввести в поле *просмотрщик* : xdvi и нажать „сохранить”.

Кроме этого, надо изменить настройки документа, включив использование SyncTex при построении dvi файла. Для этого в форме Документ → Настройки → Вывод надо взвести флаг „Synchronize with Output” и выбрать в выпадающем списке „Custom macro” значение „`\usepackage[active]{srcltx}`”.

Затем надо включить прямую связь между документом LyX и просмотрщиком dvi поместив в строку

Меню → Инструменты → Настройки...→ Вывод→ Общий→ DVI command:

значение

```
xdvi -sourceposition "$$n: $$t" $$o
```

После этого, если в панели инструментов нажата кнопка „Enable Forward/Reverse Search” то, если запущен просмотрщик dvi, то можно перемещать позицию страницы в просмотрщике в позицию курсора выбрав Меню → Навигация → Forward Search и выполнять обратную операцию с помощью комбинации Ctrl + щелчок левой кнопкой мыши.

4.3.2 Автоматическая сборка программы

Программа, создаваемая с помощью LyX и NoWeb, должна компилироваться, и, возможно, запускаться по мере создания. Данные операции не могут эффективно выполняться вручную, и, поэтому, во-первых, в документ надо включать команды для сборки, тестирования и т.п., а во-вторых LyX надо настроить так, чтобы он использовал эти включенные в текст команды.

Стартовый скрипт компиляции Следующий текст

```
#!/bin/bash
export PROJ_DIR="$1"  # скрипт сборки сможет обращаться к данным переменным
export NOWEB_SOURCE="$2"
export LYX_TMP="$3"
export NW="${LYX_TMP}/${NOWEB_SOURCE}"
export ERR_FILE="${NW}.make_err" # сообщения о ошибках запишем в данный файл
B="${NW}.build-script" # скрипт сборки
notangle -Rbuild-script "${NW}" >"$B" 2>"${ERR_FILE}"
/bin/bash "$B" >"${ERR_FILE}" 2>&1
```

надо поместить в файл $HOME/bin/build-script сделав его исполняемым с помощью команды:

```
chmod +x $HOME/bin/build-script
```

Данный скрипт инициализирует переменные окружения :
`PROJ_DIR` — с именем головной директории проекта
NOWEB_SOURCE — с именем noweb-файла, полученного из документа LyX
LYX_TMP — имя временной директории, в которой размещен документ LyX и, возможно, другие временные файлы
NW — абсолютное имя NoWeb файла.
ERR_FILE — имя файла с сообщениями о ошибках компиляции;

извлекает из документа скрипт сборки/тестирования и запускает его. При этом сообщения времени компиляции сохраняются в файле с расширением `.make_err` .

Для того, чтобы LyX стал использовать данный скрипт надо добавить в список конверторов (Меню → Инструменты → Настройки → Обработка файлов → Конверторы) конветор из *NoWeb* в *Программа* указав в поле ввода *Преобразователь*

```
build-script $$r $$i $$p
```

Теперь мы можем создавая программу указывать команды, выполнить которые нужно для сборки приложения, его запуска, проверки и т.п.

Приведем пример.

Пример использования сборки программы. Можно организовать извлечение текста файла a.cpp из документа и его компиляцию с помощью компилятора gcc

10a ⟨*a.cpp* 10a⟩≡

```
#include <stdio.h>
#include <math.h>

main(){
  printf( "############################" );   // в этой строке ошибка (умышленнаяа)
  printf( "⟨Сообщение 10b⟩\n" );
  printf( "############################\n" );
  return 0;
}
```

10b ⟨*Сообщение* 10b⟩≡ (10a)

```
Hello world
```

написав в документе

10c ⟨*build-script* 10c⟩≡ 10d ▷

```
#!/bin/bash
if [ -f a.out ] ; then rm a.out ; fi;
# извлекаем программу из документа
#notangle -Ra.cpp "${LYX_TMP}/${NOWEB_SOURCE}" >a.cpp
LNW_notangle.sh a.cpp  # Эта строка будет использована позднее
# компилируем её
g++  a.cpp
# компиляция завершена
```

Если программа успешно откомпилирована, можно её автоматически запустить.

10d ⟨*build-script* 10c⟩+≡ ◁ 10c 13f ▷

```
# запускаем программу
#./a.out
```

4.3.3 Ошибки компиляции

Компилятор выводит на экран сообщения о ошибках, используя для этого канал ошибок (`stderr`).

Скрипт `build-script` записывает сообщения о ошибках в файл с расширением `make_err`. При этом указываются номера строк, в которых обнаружены ошибки, но это строки не документа lyx а файла с текстом программы (например `a.cpp`). Формат вывода сообщения о ошибках определяется компилятором (в данном случае gcc).

Для того, чтобы LyX мог помочь в локализации ошибок выход программы должен быть представлен в формате TeX:

```
! <Краткое сообщение о ошибке>
l.200 <начало строки с ошибкой>
                                <конец строки с ошибкой>
<многострочное описание ошибки>
<пустая строка>
```

где 200 — номер строки в документе NoWeb.

Следовательно, задача заключается в том, чтобы

- связать номера строк программы на языке программирования с номерами строк документа NoWeb,
- преобразовать сообщения о ошибках к формату TeX и
- передать их в программу LyX.

Связать номера строк программы на языке программирования с номерами строк документа NoWeb можно во время извлечения текста программы из документа NoWeb, если извлечение делать не с помощью программы notangle а с помощью скрипта-оболочки `LNW_notangle.sh` (он размещен в директории `scripts`).

Скрипт `LNW_notangle.sh` в свою очередь дважды использует notangle для построения текста программы, один раз дополняя исходный текст специальными метками (`-L'#line %L "%F"%N'`). В результате работы он формирует текст программы (например файл `a.cpp`) и файл связывающий номера строк NoWeb файла с номерами строк файла программы. Данный файл имеет расширение `.nwindex` .

Скрипты `LNW_notangle.sh LNW_make_nwindex.py LNW_del_mix_nl.py`, тексты которых можно найти в директории `scripts,` необходимо поместить в каталог `$HOME/bin/` и сделать исполнимыми

```
chmod +x $HOME/bin/LNW_notangle.sh $HOME/bin/LNW_make_nwindex.py
$HOME/bin/LNW_del_mix_nl.py
```

Преобразовать сообщения о ошибках, к формату TeX и произвести подмену номеров строк можно с помощью скрипта `LNW-listerrors.` (Он тоже находится в директории `scripts`). Указать LyX, что нужно использовать данный скрипт можно указав в поле ввода *Дополнительно* конвертора из *NoWeb* в *Программа* (Меню → Инструменты → Настройки → Обработка файлов → Конверторы)

```
parselog=LNW-listerrors
```

Скрипт `LNW-listerrors` использует программу `LNW_parse_errors.py` для преобразования текста с сообщением об ошибках. (текст программы находится в директории `scripts`. Его надо скопировать в `$HOME/bin` и сделать исполняемым).

Теперь, использовать для извлечения текстов программ `LNW_notangle.sh`, то-есть в вышеприведенном примере с компиляцией `a.cpp` заменить

```
notangle -Ra.cpp "${LYX_TMP}/${NOWEB_SOURCE}" >a.cpp
```

на

```
LNW_notangle.sh a.cpp
```

то LyX автоматически укажет на обнаруженные в ходе компиляции синтаксические ошибки.

4.3.4 Ошибки извлечения текста документации

Иногда noweb не может извлечь документацию к программе из текста (сгенерировать файл TeX).

Обычно это связано с тем, что пропущены ограничители начала/конца блоков с текстом программы, опущены символы = и т.п.

Для того, чтобы локализовать ошибки, достаточно преобразовать сообщения о ошибках к формату TeX.

Для этого были написаны два скрипта. Один вызывает программу `noweave` сохраняя сообщения о ошибках, а второй — преобразовывает сохраненные сообщения к формату TeX.

Для того, чтобы данные скрипты начали использоваться надо

- сохранить скрипты `LNW_noweave.sh` и `LNW_noweave_parse_errors.sh` в директории `$HOME/bin` (их можно взять в директории `scripts`) и сделать их исполнимыми
  chmod +x `$HOME/bin/LNW_noweave.sh $HOME/bin/LNW_noweave_parse_errors.sh`
- настроить конверторы из *NoWeb* в *LaTeX (pdflatex)*и в *LaTeX (plain)* указав в поле *конвертер*
  `LNW_noweave.sh $$i $$o`
  а в поле *Преобразователь* —-
  `parselog=LNW_noweave_parse_errors.sh`

4.3.5 LyX и система контроля версий SVN

Добавляем новый проект в репозиторий svn (созданный ранее командой `svnadmin create /path/to/project`)

```
svn import .  \
svn+ssh://SD@icecube.satellite.dvo.ru/home/SD/Runx/svn/repos/new_project \
-m "First Import"
```

Получение начальной версии на свой компьютер (создание локальной версии) в текущую директорию

```
svn checkout \
svn+ssh://SD@icecube.satellite.dvo.ru/home/SD/Runx/svn/repos/new_project
```

Обновить до текущей рабочей версии
`svn update`

4.4 Работа с программой LyX для набора текстов

Работа с программой LyX для набора обычных текстов (включающих оглавление, изображения и таблицы) не является сложной. Использование её позволяет достичь высокого качества набора, хотя для достижения эксклюзивных результатов придется обратится к специалистам по системе TeX и литературе. При наборе текста в LyX (как и в некоторых других программах) лучше не концентрироваться на внешнем виде документа, а, вместо этого, сконцентрироваться на его структуре. Как правило выбор правильной структуры гарантирует хороший внешний вид, который можно улучшить позднее, после того как содержание документа будет полностью готово.

Прекрасным введением в работу является переведенный Максимом Дзюманенко на русский язык самоучитель, включенный в поставку программы.

5 Примеры создания программы в системе LyX/NoWeb

5.1 Программа выводящая простые числа от a до b

Программа вывода простых чисел организована следующим образом

13a ⟨*ядро программы* 13a⟩≡ (13e)

```
for( int i = a; i <= b; i++ ){
   if( ⟨i - простое число 13c⟩ ){
      printf( "%d - is prime number\n", i );
   }
}
```

Проверку того, является ли число простым, можно вынести в отдельную функцию:

13b ⟨*функции* 13b⟩≡ (13e)

```
int is_prime( int n );
```

Тогда проверка будет выглядеть следующим образом:

13c ⟨*i - простое число* 13c⟩≡ (13a)

```
is_prime(i)
```

Определим функцию проверки, используя перебор, при этом должны нассматриваться делители от 2 до $\sqrt{n}$:

13d ⟨*опр функций* 13d⟩≡ (13e)

```
int is_prime( int n ){
   for( int j = 2; j < n; j++ ){
      if( n % j == 0 ) return 0;
   }
   return 1;
}
```

Соберем программу в файл

13e ⟨*prime.c* 13e⟩≡

```
#include <stdio.h>

⟨функции 13b⟩
⟨опр функций 13d⟩


main(){
   int a = 2;
   int b = 20;
   ⟨ядро программы 13a⟩
}
```

Компиляция и запуск программы:

13f ⟨*build-script* 10c⟩+≡ ◁10d 18▷

```
LNW_notangle.sh prime.c
# компилируем её
g++  prime.c -o prime
./prime
```

5.2 Поиск оптимальной позиции начала выброса в длинных нардах

5.2.1 Постановка задачи

В игре „длинные нарды" играющие должны сначала вывести свои 15 фишек на последние 6 позиций игрового поля, в „дом", а затем „выбросить" их с доски. Тот, кто выбросит фишки первым — выигрывает.

Ход, который может сделать игрок определяется случайным образом, с помощью двух костей. Если игроку выпали, например, 3 и 4, то игрок может сходить на 3 и 4, а может на 4 и 3. Если выпадает два одинаковых значения — „дубль", пример 3 3, то игрок ходит не два а четыре раза — 3, 3, 3, 3.ru

Пронумеруем позиции игрового поля числами с 5 до 0. Пусть на каждой позиции a_i фишек (в начале игры $\sum a_i = 15$). Тогда одиночный ход на L позиций ($L \in [1,6]$) это перенос фишки в с позиции i на позицию $j = i - L + 1$. Если эта позиция находится за пределами доски ($j < 0$) то фишка выбрасывается если выполняется одно из двух условий: $j = -1$ или $\sum_{j=(i+1)\ldots 5} a_i = 0$.

Вопрос, какое среднее число бросков потребуется игроку, чтобы „выбросить" все свои фишки с доски?

Какое положение фишек является оптимальным для начала выброса?

5.2.2 Схема решения

Пусть есть позиция, заданная в виде массива из шести элементов, для которой мы должны определить среднее количество оставшихся бросков.

Если позиция пустая, то ответ очевиден — ноль.

Если позиция не пустая мы должны просмотреть комбинации костей, которые могут выпасть, для каждой из них найти лучшие ходы фишек.

$$S(pos) = \begin{cases} pos = 0: & 0 \\ pos \neq 0: & \frac{1}{36} \sum_{k_1 \in [1,6],\, k_2 \in [1,6]} \left(1 + \min_{step \in MS(pos,k_1,k_2)} S\left(Next\left(pos, step\right)\right)\right) \end{cases}$$

здесь MS — множество возможных ходов из позиции pos если выпали кости k_1, k_2, а $Next$ — позиция в которую переходит игра после шага $step$ из позиции pos.

Данную задачу можно решить численно с помощью рекурсивного вычисления значения функции $S(pos)$. В ходе решения придется многократно находить значения функции для одних и тех-же позиция, поэтому необходимо сохранять уже найденные решения в словаре.

5.2.3 Текст программы

Программу напишем на языке `python`.

Позицию будем описывать как массив из шести элементов, по числу позиций „дома". В элементе массива будет содержаться число фишек, находящихся в данной позиции или ноль, если позиция пуста.

Тогда функция `S` будет выглядеть следующим образом:

14 ⟨*пример. функции* 14⟩≡ 15c ▷

```
slov = {}
def S(pos):
   global slov
   if ⟨pos - конечная позиция 15b⟩: return 0
   h = ⟨индекс позиции pos в словаре 15a⟩
   if h in slov : return slov[h]
   res = 0.0;
   for k1 in [1,2,3,4,5,6] :
     for k2 in [1,2,3,4,5,6] :
        pos2 = ⟨результат лучшего хода из позиции pos, для броска k1,k2 16b⟩
```

```
            # print 'best action:', pos, k1, k2, pos2
            res += S(pos2) + 1
        res = res / 36.;
        slov[h] = res
        return res
```

Индексы для кеша позиций должны быть уникальными целыми числами, при этом на одной позиции не может находиться более 15 фишек одновременно.

Поэтому:

15a ⟨*индекс позиции pos в словаре* 15a⟩≡ (14)

```
sum( pos[i]*(16**i) for i in [0,1,2,3,4,5] )
```

А проверку того, является ли позиция конечной целью игры можно выполнить так:

15b ⟨*pos - конечная позиция* 15b⟩≡ (14 15c)

```
0 == sum( pos )
```

Перейдем к определению функции `SM`, которая должна возвратить список `позиций` в которые могут перейти фишки если на костях выпало `k1`, `k2`.

Рассмотрим сначала более простую задачу, при которой надо рассмотреть бросок не пары, а одной кости. При этом если бросок производится из конечной позиции игры, мы конечную позицию и возвращаем, считая что игра закончилась при игре предыдущей костью.

15c ⟨*пример. функции* 14⟩+≡ ◁14 16a▷

```
def SM1( pos, k ):
    """список позиций, в которые может перейти игра из позиции pos если выпало k"""
    # заносим список позиций, в которые может перейти игра, в res
    if ⟨pos - конечная позиция 15b⟩ :
       # если pos - конечная позиция то игра закончена
       #  возвращаем конечную позицию
       yield pos
       return
    else :
       for i in [0,1,2,3,4,5]:
          if pos[i] == 0: continue
          pos2 = pos[:]
          pos2[i] -= 1
          j = i - k
          if j >= 0 :
              pos2[j] += 1 # обычный ход
          else :
              # выброс
              if j != -1 :
                  # позиция фишки не соответствует выпавшему камню,
                  # в этом случае выброс производится только с крайней позиции
                  if sum( pos[i+1:] ) != 0:
                      continue  # позиция не является крайней
          yield pos2   # возвращаем результат
```

Тогда полный список позиций можно получить следующим образом

16a ⟨*пример. функции* 14⟩+≡ ◁15c

```
def SM( pos, k1, k2 ):
    """список позиций, в которые может перейти игра
       из позиции pos если выпало k1 и k2 (с учетом дублей)"""
    res = []
    if k1 != k2 : # не выпал дубль
       for pos1 in SM1( pos, k1 ):
          for pos2 in SM1( pos1, k2 ):
              res += [pos2]
       for pos1 in SM1( pos, k2 ):
          for pos2 in SM1( pos1, k1 ):
              res += [pos2]
    else : # выпал дубль
      for pos1 in SM1( pos, k1 ):
        for pos2 in SM1( pos1, k1 ):
          for pos3 in SM1( pos2, k1 ):
            for pos4 in SM1( pos3, k1 ):
               res += [pos4]

    # удаляем дубликаты конечных позиций
    res.sort()
    res2 = []
    i = 0
    for r in res:
      if i == 0 or (i>0 and res[i] != res[i-1] ):
         res2 += [r]
      i += 1
    return  res2
```

Вычисление значения ***результат лучшего хода из позиции pos, для броска k1, k2*** лучше вынести в отдельную функцию. Тогда

16b ⟨*результат лучшего хода из позиции pos, для броска k1,k2* 16b⟩≡ (14)

```
best_target_for( pos, k1, k2 )
```

16c ⟨*пример. функции* 16c⟩≡ (17d)

```
def best_target_for( pos, k1, k2 ):
    best_S = 9999;
    best_pos = [];
    for pos2 in SM( pos, k1, k2 ):
       a = S(pos2)
       if a < best_S :
           best_S = a
           best_pos = pos2[:]
return best_pos
```

Получим ответ на вопрос, какое среднее количество бросков надо сделать игроку из указанной позиции для завершения игры.

17a ⟨*ответ на вопрос 1* 17a⟩≡ (17d)

```
a = [0,1,10,1,3,0] # начальная позиция игры
res = S(a)
print "Для позиции " + repr(a),
print     "Среднее количество бросков до завершения игры равно " + str(res)
```

Для ответа на второй вопрос (о оптимальной позиции начала выброса) сначала надо получить набор всех начальных позиций.

17b ⟨*start _pos <- набор начальных позиций* 17b⟩≡

```
startpos = []
for p1 in range(0, 15+1):
  for p2 in range(p1, 15+1):
    for p3 in range(p2, 15+1):
      for p4 in range(p3, 15+1):
        for p5 in range(p4, 15+1):
          pos = [p1, p2-p1, p3-p2, p4-p3, p5-p4, 15-p5 ]
          start_pos += [pos]
```

После этого мы может получить ответ на второй вопрос, учитывая про этом среднее количество бросков, которое дополнительно потребовалось игроку, чтобы занять начальную позицию.

17c ⟨*ответ на вопрос 2* 17c⟩≡ (17d)

⟨*start_pos <- набор начальных позиций* (never defined)⟩

```
for pos in start_pos:
    rolls_to_start_pos = (
         (pos[0]*5+pos[1]*4+pos[2]*3+pos[3]*2+pos[4]*1+pos[5]*0)/8.166666666
                        )
    rolls_to_exit = S(pos)
    full_rolls = rolls_to_exit + rolls_to_start_pos
    print "%6.3f" % full_rolls, "%25s" % repr(pos) ,
    print  "%6.3f" % rolls_to_start_pos, "%6.3f" % rolls_to_exit
```

Соберем программу.

17d ⟨*nardy.py* 17d⟩≡

```
#!/usr/bin/python
# -*- coding: utf8 -*-
import sys

⟨пример. функции 16c⟩

def main( args ):
    ⟨ответ на вопрос 1 17a⟩

    ⟨ответ на вопрос 2 17c⟩

if __name__ == "__main__" :
    main( sys.argv )
```

и добавим в скрипт сборки строки для её извлечения из документа и проверки.

18 ⟨*build-script* 10c⟩+≡ ◁13f

```
#!/bin/bash

LNW_notangle.sh -x nardy.py "$PROJ_DIR" # Эта строка будет использована позднее
cd "$PROJ_DIR"
pycompile nardy.py
```

Программа написана.

Обнаруженная оптимальная позиция для начала выброса — `[0, 0, 0, 3, 5, 7 ]`.

6 Заключение

Созданная система грамотного программирования использоуется с лаборатории спутникового мониторинга ИАПУ ДВО РАН.

Опыт ее использования показал, что она действительно помогает создавать хорошие, полезные и долгоживущие программы.