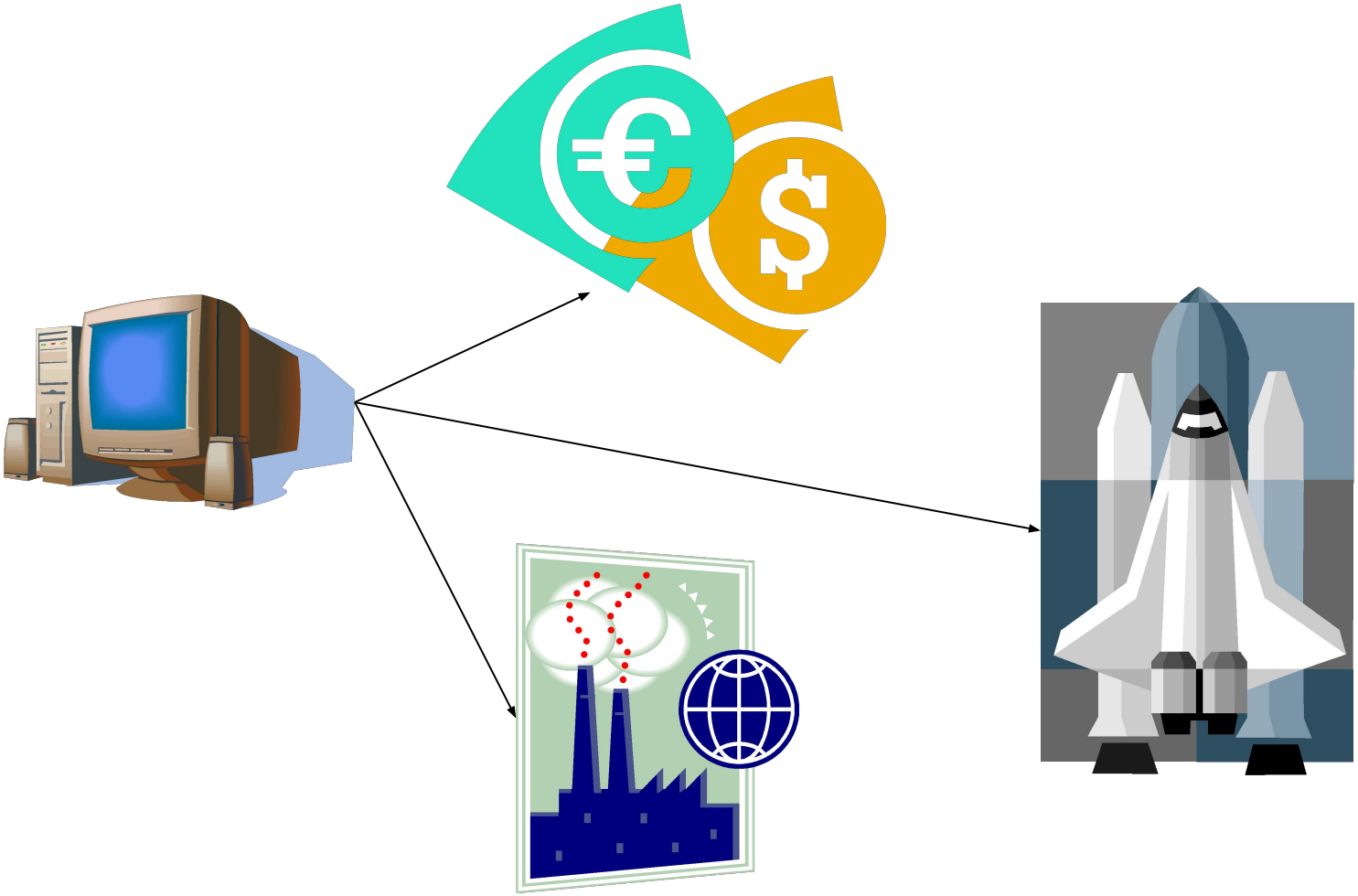
Технология программирования

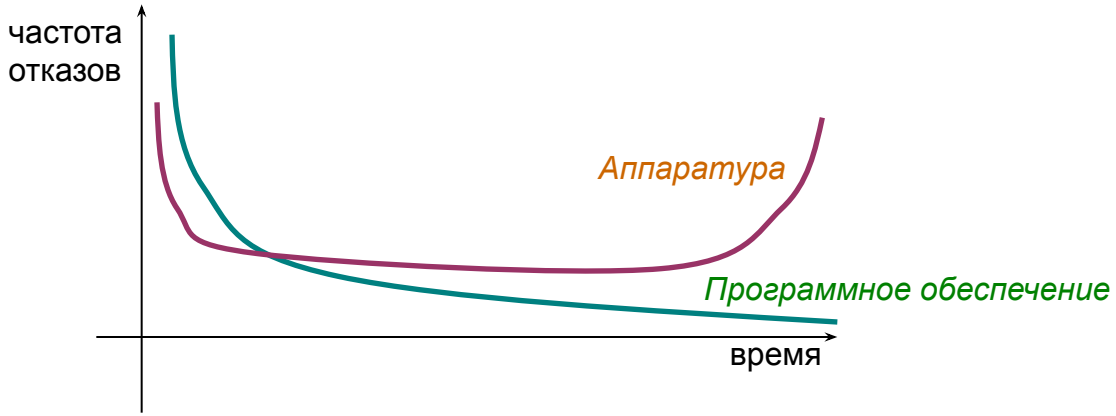
Надежность программного обеспечения

Введение



Введение

- **Ошибка** – это неспособность системы действовать в соответствии с перечнем требований, предъявляемых к системе
- **Ошибка** имеет место, если программное обеспечение не выполняет того, что пользователю разумно от него ожидать
- **Отказ** – проявление ошибки



Введение

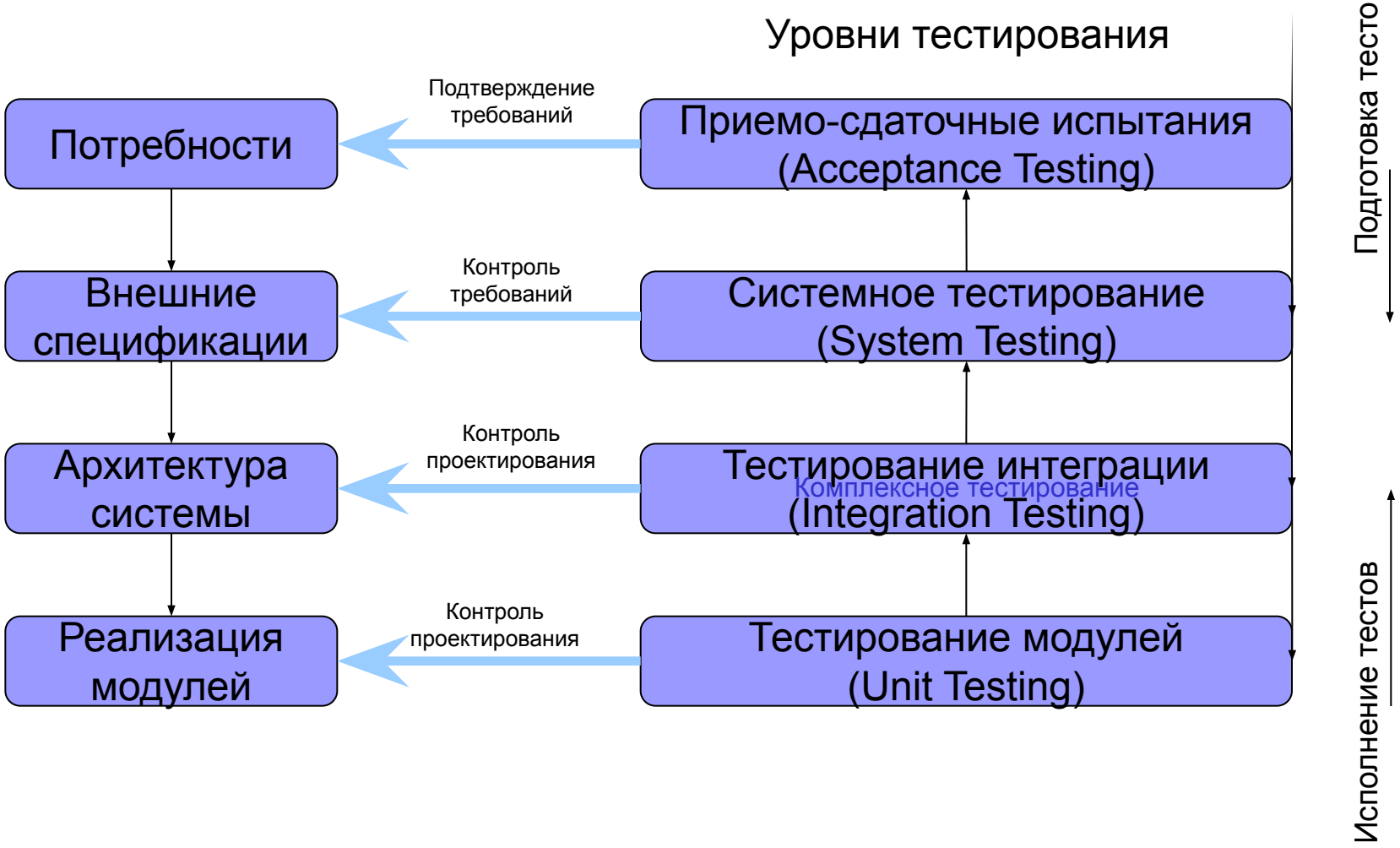
- **Тестирование** – это процесс выполнения программы с целью обнаружения ошибок
- **Доказательство** – попытка найти ошибку в программе безотносительно к входным данным

- **Контроль (верификация)** – поиск ошибок в ПО при выполнении его в тестовой моделируемой среде
- **Испытание** – поиск ошибок в ПО при выполнении его в заданной реальной среде

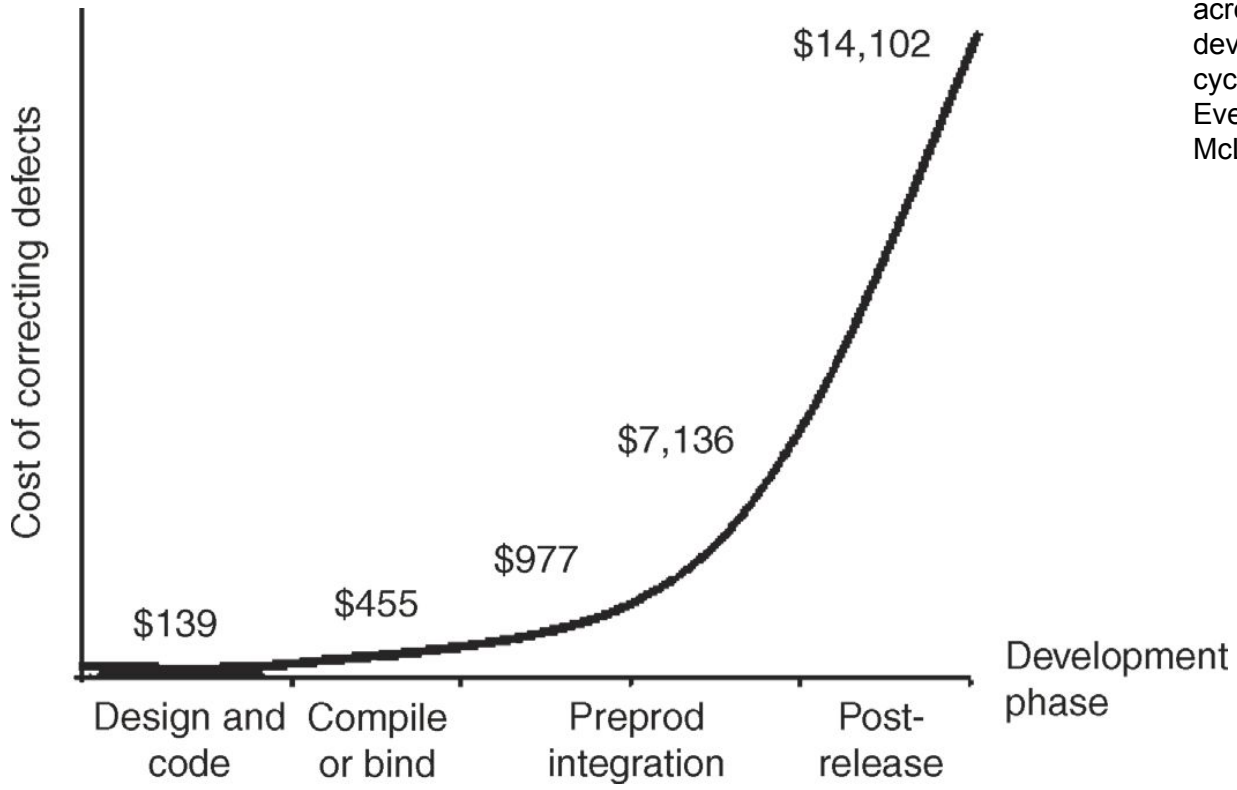
- **Отладка** – установление точной причины обнаруженной ошибки. Результаты тестирования являются исходным материалом для отладки

- **Надежность ПО** – это вероятность его работы без отказов в течение определенного периода времени, рассчитанная с учетом стоимости для пользователя каждого отказа

Связь процесса тестирования и процесса проектирования



Связь процесса тестирования и процесса проектирования



Defect correction cost profile for the software industry

Software testing : testing across the entire software development life cycle / by Gerald D. Everett, Raymond McLeod, Jr.

Уровни тестирования

- **Тестирование модуля.** Выявление ошибок в минимальных элементах программной системы. Выполняется по мере разработки модулей
- **Интегрированное тестирование.** Тестируется взаимодействие между компонентами, в том числе и сторонними
- **Системное тестирование.** Тестирование разработанной системы в целом. Цель проверить, что все системные элементы объединены и выполняют заданные функции. Может выполняться в моделируемой или реальной среде
- **Приемо-сдаточное тестирование.** Проверка готовности для использования конечными пользователями. Цель – подтвердить, что функции, описанные в спецификациях соответствуют ожиданиям пользователей. Альфа-тестирование – заказчиком в организации разработчика. Бета-тестирование – выполняется конечными пользователями.

Виды тестирования

- **Тестирование функциональности** (functionality testing)
  - Функциональные тесты (function test). Выявляют ошибки в реализации требуемых функций. Эти тесты выполняются для модулей, интеграции компонентов, приложений и систем в целом
  - Тесты безопасности (security test). Проверка того, что данные и функции системы доступны только тем актантам, которым они предназначены. Выполняются на всех уровнях тестирования
  - Тесты предельных значений (volume test). Тестируется возможность обработки максимальных объемов данных
- **Тестирование практичности/удобства** (usability testing). Включает тестирование удобства пользовательского интерфейса, всех видов эксплуатационной документации (включая оперативную документацию)
- **Тестирование надежности** (reliability testing)
  - Тесты интеграции (integrity). Дают оценку устойчивости к ошибкам отдельных модулей и их сборок
  - Тесты структуры (structure). Например, в веб-приложении проверяется корректность всех ссылок, определяется правильность предоставления информации
  - Стрессовые (stress) тесты. Проверяется работоспособность приложения в «ненормальных» условиях (превышение загрузки, недостаток памяти, недоступные устройства, недоступные внешние компоненты)
- **Тестирование производительности** (performance testing)
  - Сравнительные (benchmark) тесты. Сравнение производительности разрабатываемой системы со сторонними системами
  - Тесты конфликтов (contention). Тестирование одновременного доступа актантов к одним и тем же ресурсам (БД, память, и т.д.)
  - Нагрузочные (load) тесты. Тестирование границ приемлемого выполнения функций под изменяющейся нагрузкой. Эмулируются средние и пиковые нагрузки. Анализируется время ответа и время реакции системы на запросы (реакция системы – некоторые ответные действия системы, не обязательно ответ на запрос).
- **Тестирование сопровождаемости** (supportability testing)
  - Тесты конфигурирования (configuration). Проверяется правильность функционирования при различных конфигурациях программного обеспечения и аппаратуры.
  - Тесты инсталляции (installation). Тесты возможности инсталляции системы при различных конфигурациях программного обеспечения и аппаратуры.

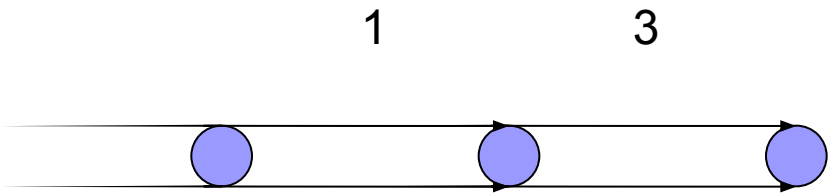
Стратегии тестирования

Учет сведений о внутренней реализации

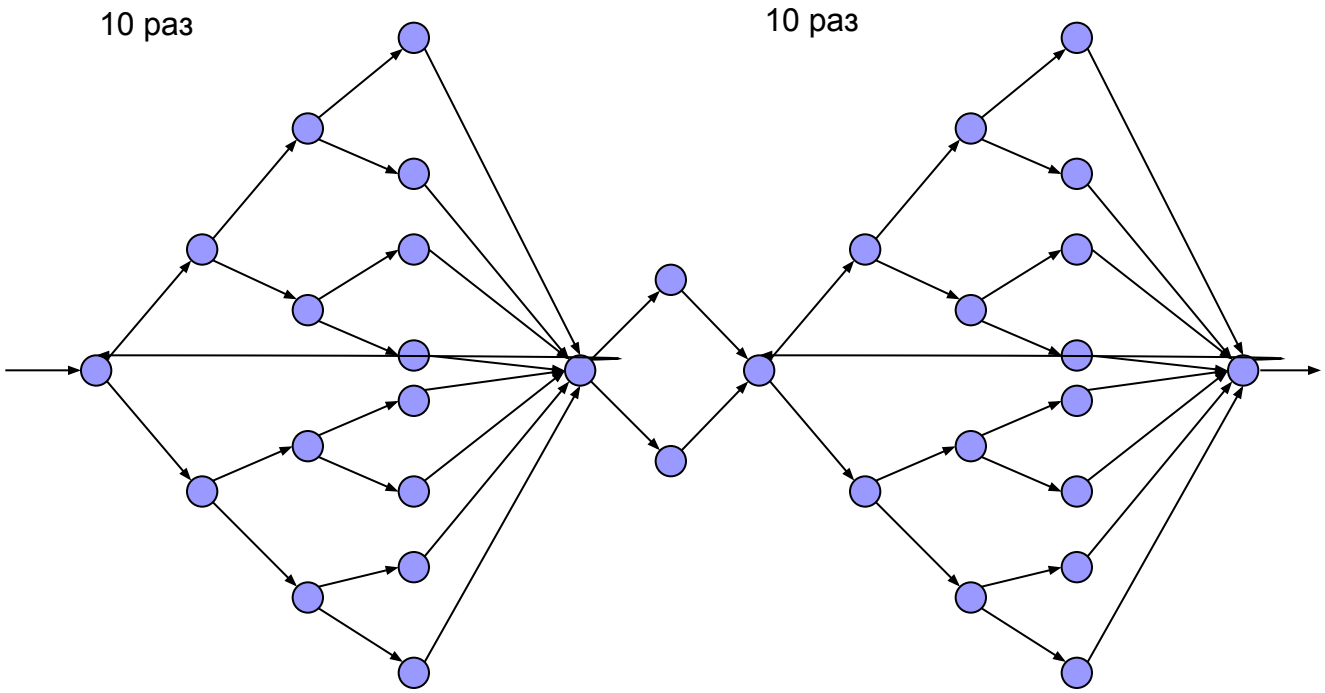
Тестирование по спецификациям
Тестирование «черного ящика»
Формальное тестирование

Тестирование по тексту программ
Тестирование «белого ящика»
Содержательное тестирование

Стратегии тестирования



4 теста

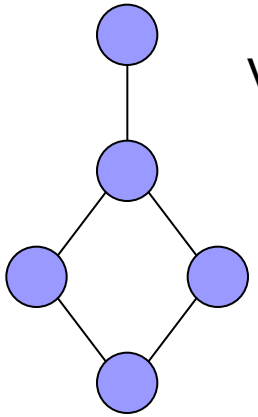


$N = (2^3)^{10} \cdot 2 \cdot (2^3)^{10} = 2^{61} = 2 \cdot (2^{10})^6 > 2 \cdot (10^3)^6 = 2 \cdot 10^{18}$

Стратегии тестирования

Цикломатическая сложность процедуры – количество независимых путей

V(G)=1



V(G)=2

…

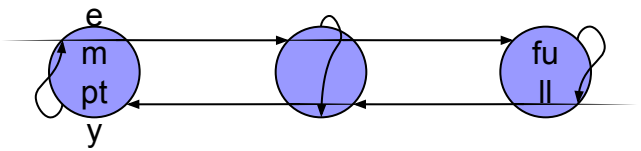
V(G)=P+1

P – количество ветвлений

Классы эквивалентности и граничные значения

Количество студентов [0;70]

Стек



Тестирование модуля

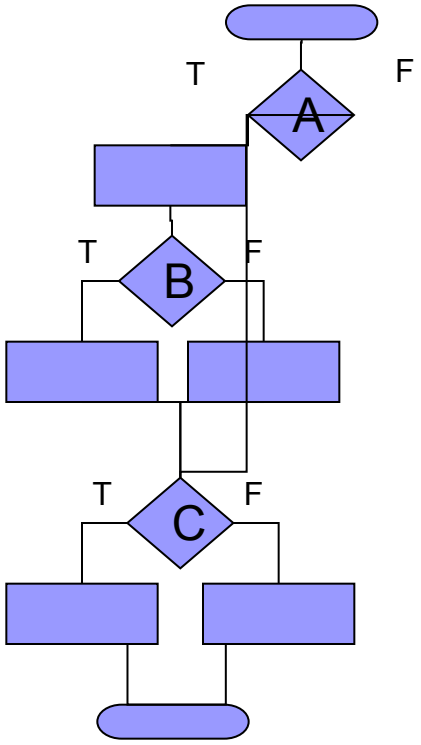
1. Тестирование модуля, как черного ящика

Матрица тестирования
классов эквивалентности и граничных значений

Входные данные и ожидаемые результаты	Номера тестов			
	1	2	3	4
1)	X		X	
2)		X	X	
3)	X			
4)			X	X
5)				X

Тестирование модуля

2. Тестирование базового пути (каждая ветвь хотя бы один раз)



Матрица учета ветвей

Ветвления	Ветви	Номера тестов					
		1	2	3	4	5	6
A	T	X		X		X	X
	F		X		X		
B	T	X		X			
	F					X	X
C	T	X	X	X	X	X	
	F						X

Ранее подготовленные тесты

Дополнительные тесты

Тестирование модуля

3. Тестирование циклов (в т.ч. вложенных)

Матрица учета циклов

Цикл	Количество итераций	Номера тестов							
		1	2	3	4	5	6	7	8
L	0							X	
	1	X				X			
	максимальное								X

Ранее подготовленные тесты

Дополнительные тесты

4. Тестирование чувствительности к входным данным

Это тестирование граничных значений локальных структур данных:

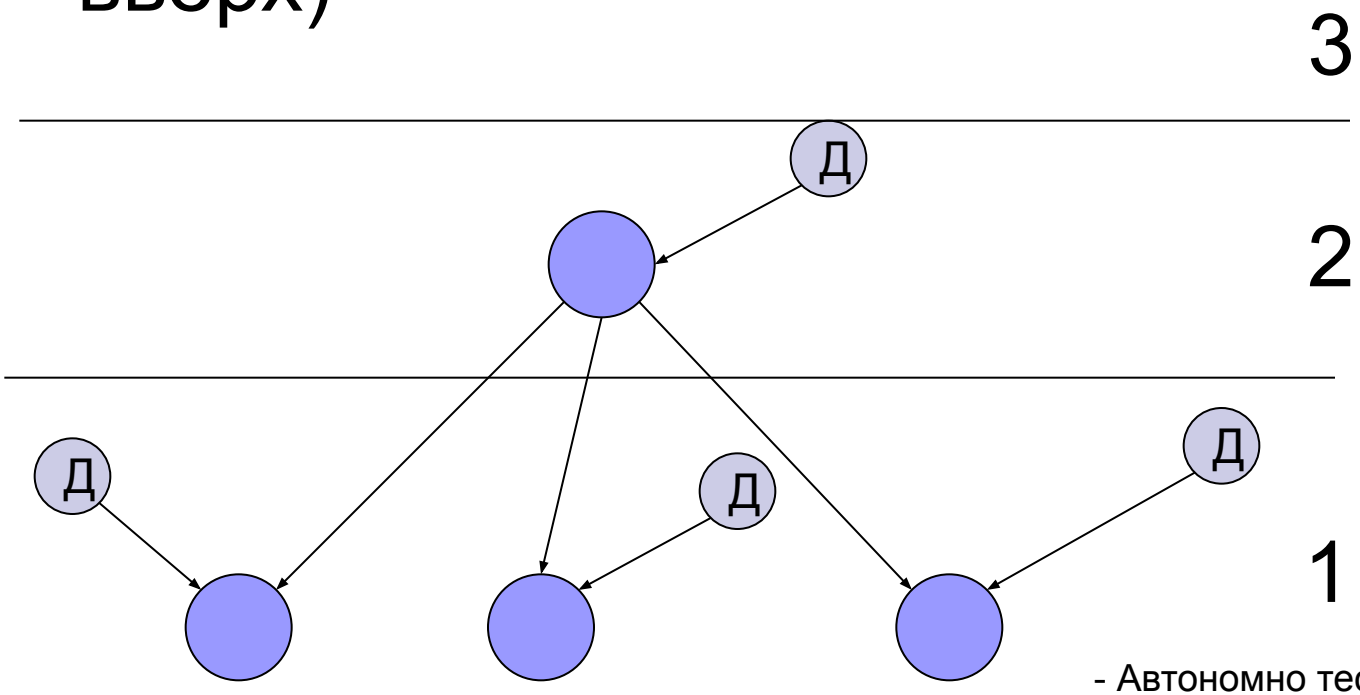
- ситуации деления на ноль, переполнение
- утечки памяти

Тестирование интеграции (интегрированное тестирование)

- Цель – обнаружение ошибок взаимодействия модулей
- Последовательность тестирования определяется порядком сборки (интеграции) модулей
- Средства: драйверы и заглушки

Тестирование интеграции

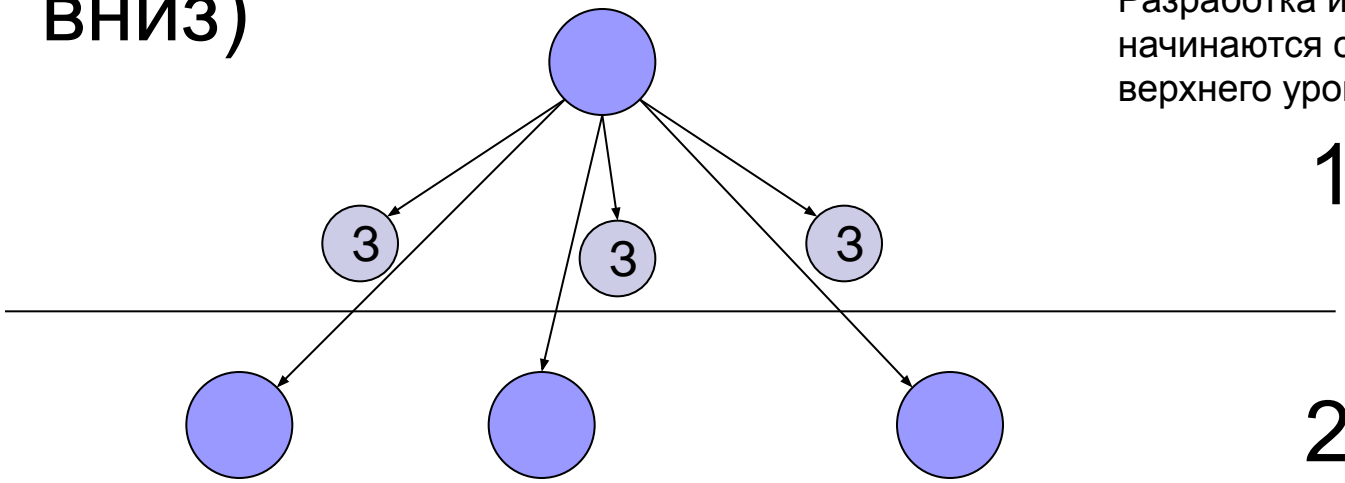
- Восходящее тестирование (снизу - вверх)



- Автономно тестируются только модули нижнего уровня
- Количество драйверов определяется количеством модулей
- Ошибки локализуются в последнем подключаемом модуле

Тестирование интеграции

- Нисходящее тестирование (сверху - вниз)



Разработка и тестирование начинаются с модуля верхнего уровня

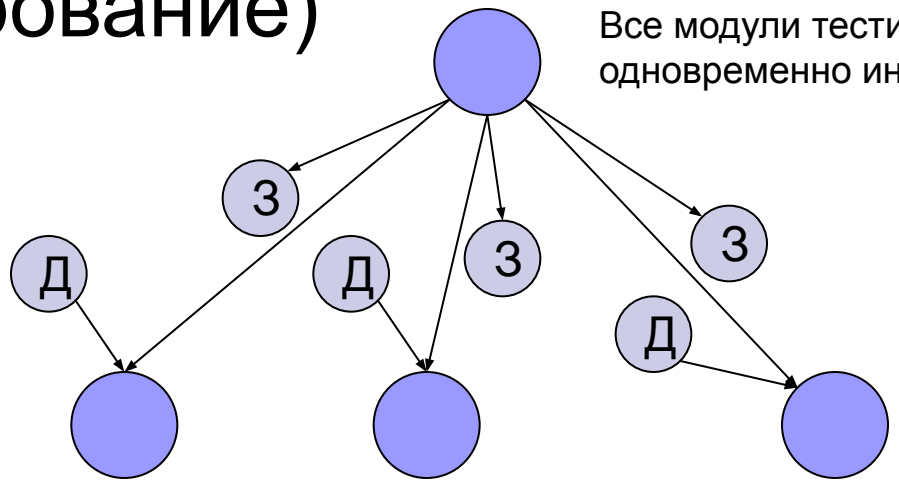
Проблемы:
- Обращение к модулю, который еще не существует
- Передача тестовых данных модулю самого верхнего уровня
- Сложные заглушки с тестовыми данными

Достоинства:
- Совмещение тестирования модулей, интеграции и системное тестирование (функциональное) совмещены и выполняются на ранних этапах
- Тестовые данные готовятся в естественном виде (при наличии модулей ввода-вывода)
- Меньшее количество данных (ограничение передачи данных вниз)

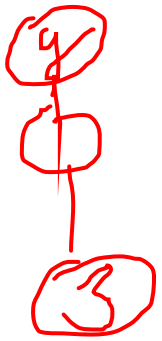
Недостатки:
- Большое количество отложенных решений
-Низкая надежность модулей нижних уровней

Тестирование интеграции

- Метод «большого скачка» (монолитное тестирование)



Все модули тестируются автономно и одновременно интегрируются в систему



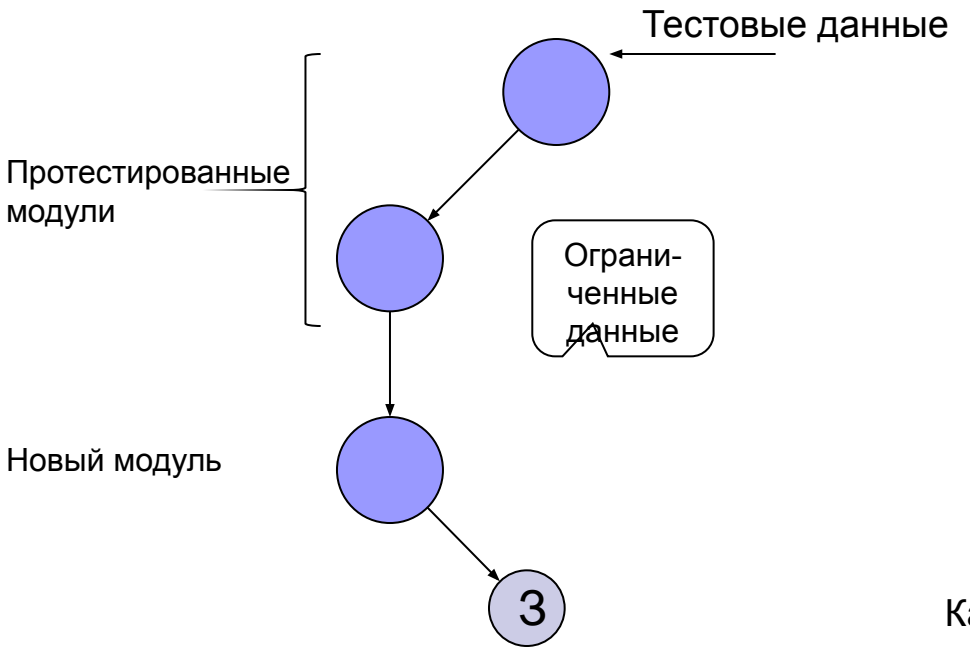
Недостатки:
- Необходимость и в драйверах, и в заглушках
- Модули долгое время не тестируются совместно
- Трудность локализации ошибок

Достоинства:
- Распараллеливание работ по проектированию и автономному тестированию

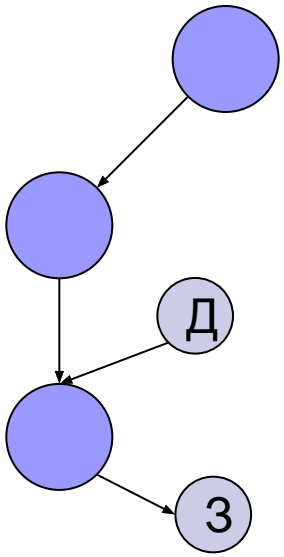
Тестирование интеграции

- Модифицированный нисходящий

В нисходящем методе



Модификация

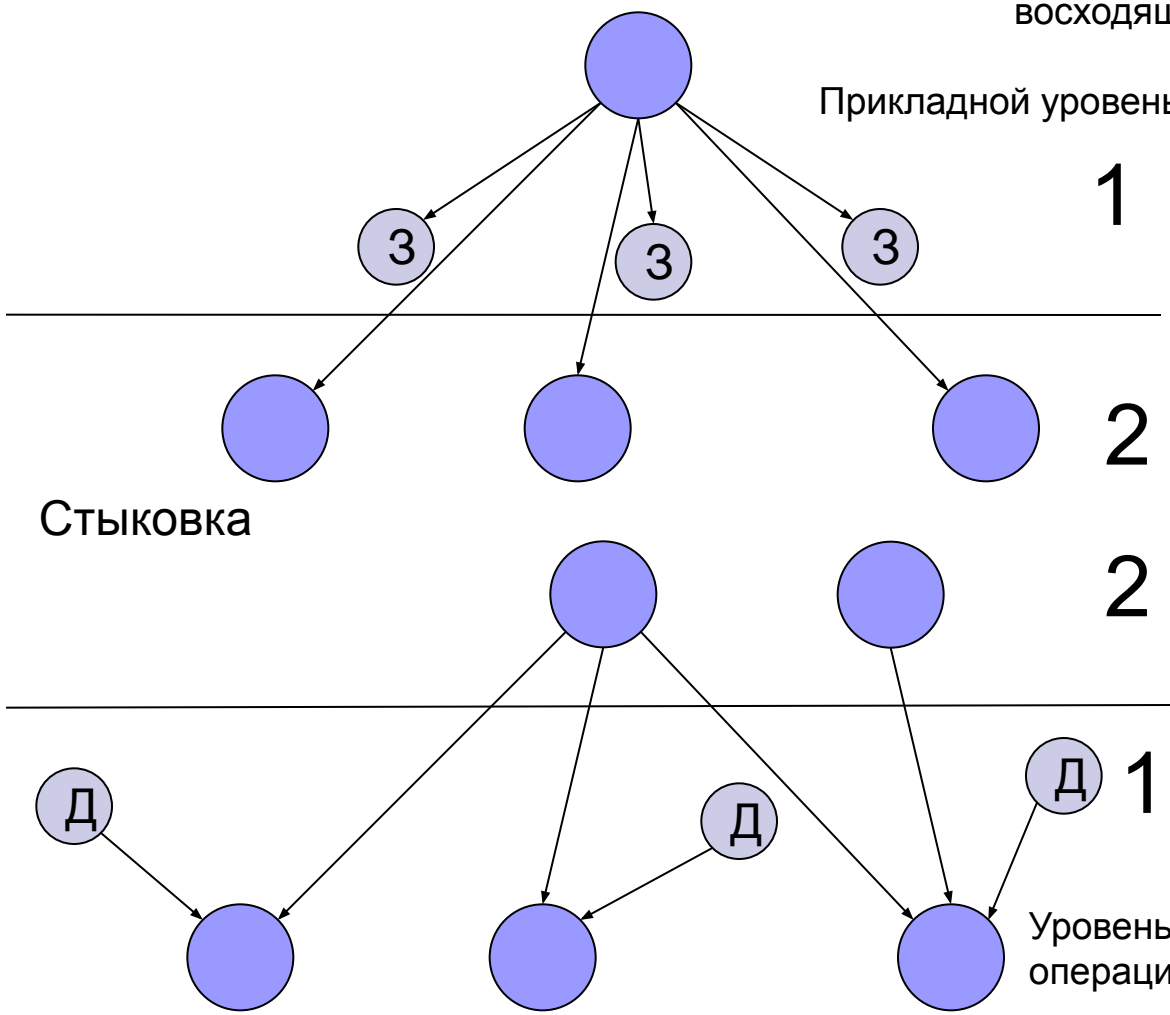


Каждый модуль проходит автономное тестирование перед интеграцией в систему

Тестирование интеграции

- Метод сэндвича

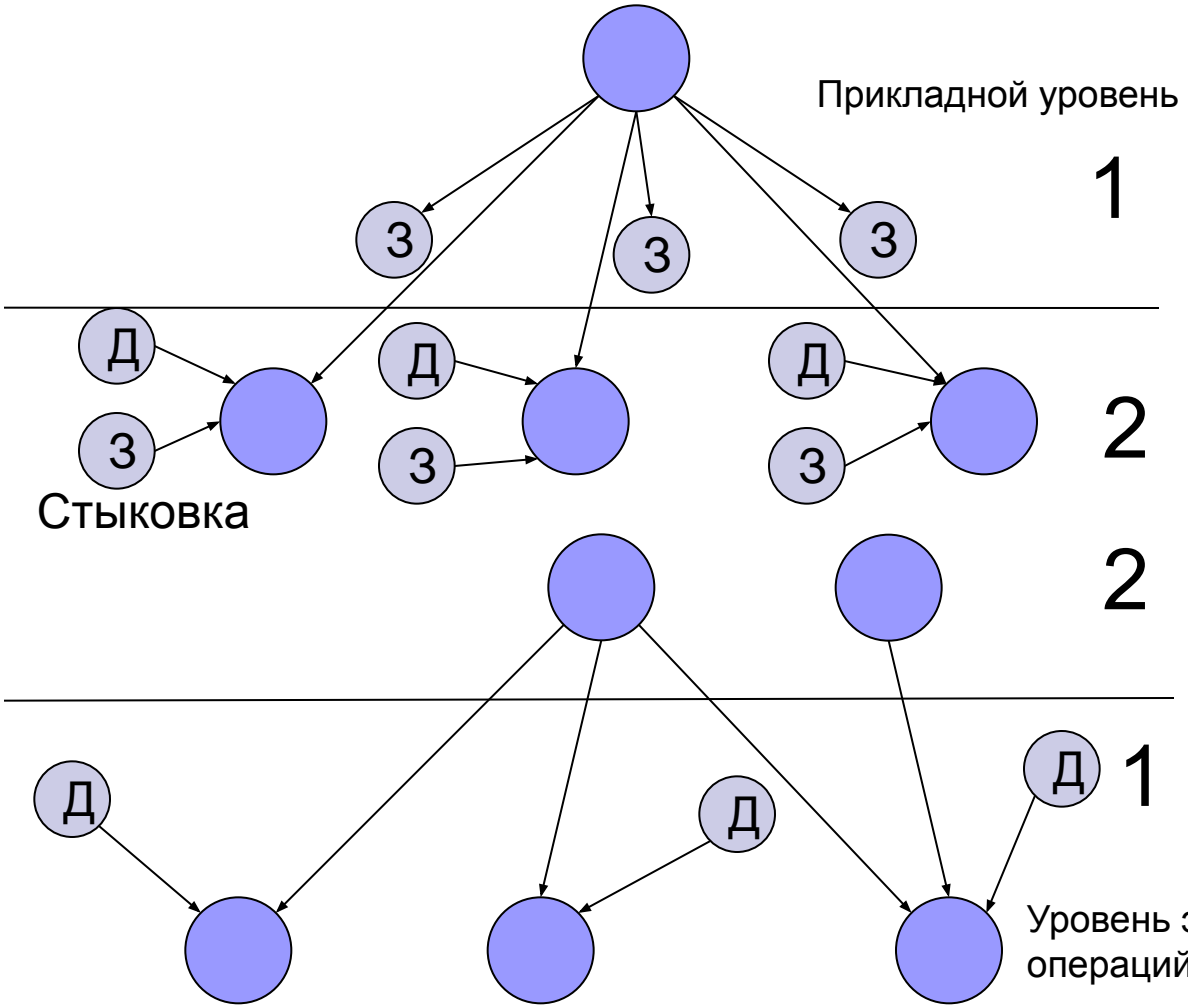
Одновременно начинается нисходящая и восходящая интеграция и тестирование.



Достоинства:
- Раннее начало интеграции системы и тестирования
- Надежное тестирование модулей нижних уровней

Тестирование интеграции

- Модифицированный метод сэндвича



То же самое, что метод сэндвича, но модули верхнего уровня тестируются сначала автономно (модифицированный нисходящий метод). Это требует дополнительных драйверов

Тестирование интеграции

Сравнительный анализ

Показатель	Восхо-дящий	Нисхо-дящий	«Большо-го скачка»	Модиф. нисхо-дящий	Сэндвича	Модиф. сэндвича
Время сборки (обнаружение ошибок в интерфейсах)	Рано (+)	Рано (+)	Поздно (-)	Рано (+)	Рано (+)	Рано (+)
Время создания работающего варианта (обнаружение принципиальных ошибок)	Поздно (-)	Рано (+)	Поздно (-)	Рано (+)	Рано (+)	Рано (+)
Необходимость в драйверах	Да (-)	Нет (+)	Да (-)	Да (-)	Частично (±)	Да (-)
Необходимость в заглушках	Нет (+)	Да (-)	Да (-)	Да (-)	Частично (±)	Частично (±)
Параллельность в начале работы (использование кадровых ресурсов)	Средняя (±)	Слабая (-)	Высокая (+)	Средняя (±)	Средняя (±)	Высокая (+)
Возможность тестирования отдельных путей (полнота тестирования)	Легко (+)	Трудно (-)	Легко (+)	Легко (+)	Средняя (±)	Легко (+)
Возможность планирования и контроль процесса тестирования	Легко (+)	Трудно (-)	Легко (+)	Трудно (-)	Трудно (-)	Трудно (-)
Оценка в баллах	**9**	2	7	0	7	**10**

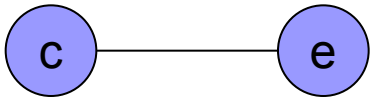
Системное тестирование (разработка функциональных тестов)

Метод диаграмм причин-следствий (cause-effect):

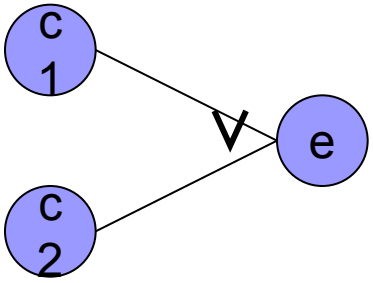
1. Выявляются **причины** (значения из классов эквивалентности входных данных) и **следствия** (ожидаемые отклики системы)
2. Разрабатывается граф причинно-следственных связей (автоматная модель приложения)
3. Формируется таблица решений
4. Столбцы решений образуют тестовые данные (тестовые случаи)

Метод диаграмм причин-следствий

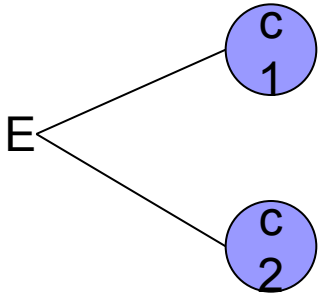
c = {0,1} e = {0,1}



Функция «тождество»

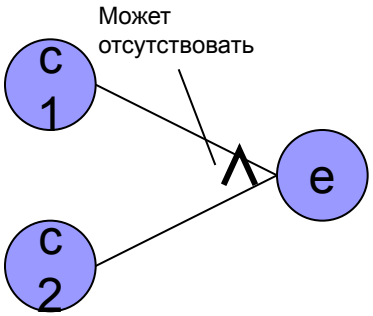


Функция «или»

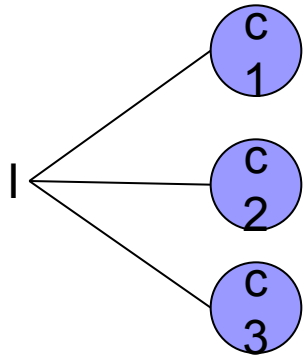


Ограничение «исключает»
(только одна величина = 1)

Функция «не»



Функция «и»



Ограничение «включает»
(хотя бы одна величина = 1)

Метод диаграмм причин-следствий

Пример. *Тестировать исполнение команды операционной системы*

rename name1 [name2]

Описание тестируемой функции

- Заменить имя name1 указанного файла на name2
- Если не указано name2, то файл с name1 удаляется
- Сообщать об ошибке, если нет файла с name1
- Сообщать об ошибке, если нарушен синтаксис команды
- Если файл с name2 существует, то не удалять его, и сообщать об ошибке

Метод диаграмм причин-следствий

Причины:

- (c1) Длина name1 от 1 до 8 символов
- (c2) Длина name2 от 1 до 8 символов
- (c3) Длина name2 равна 0
- (c4) Файл с именем name1 существует
- (c5) Файл с именем name2 существует

rename name1 [name2]

Полный перебор требует $2^5 = 32$ теста

Промежуточные причины:

- (c6) Правильные name1 и name2
- (c7) Правильное name1, а name2 отсутствует
- (c8) Команда синтаксически правильная

Следствия:

- (e1) Файл переименован
- (e2) Сообщение «Файл переименован»
- (e3) Сообщение «Нет файла»
- (e4) Сообщение «Синтаксическая ошибка»
- (e5) Файл удален
- (e6) Сообщение «Файл удален»
- (e7) Сообщение «Файл существует»

Метод диаграмм причин-следствий

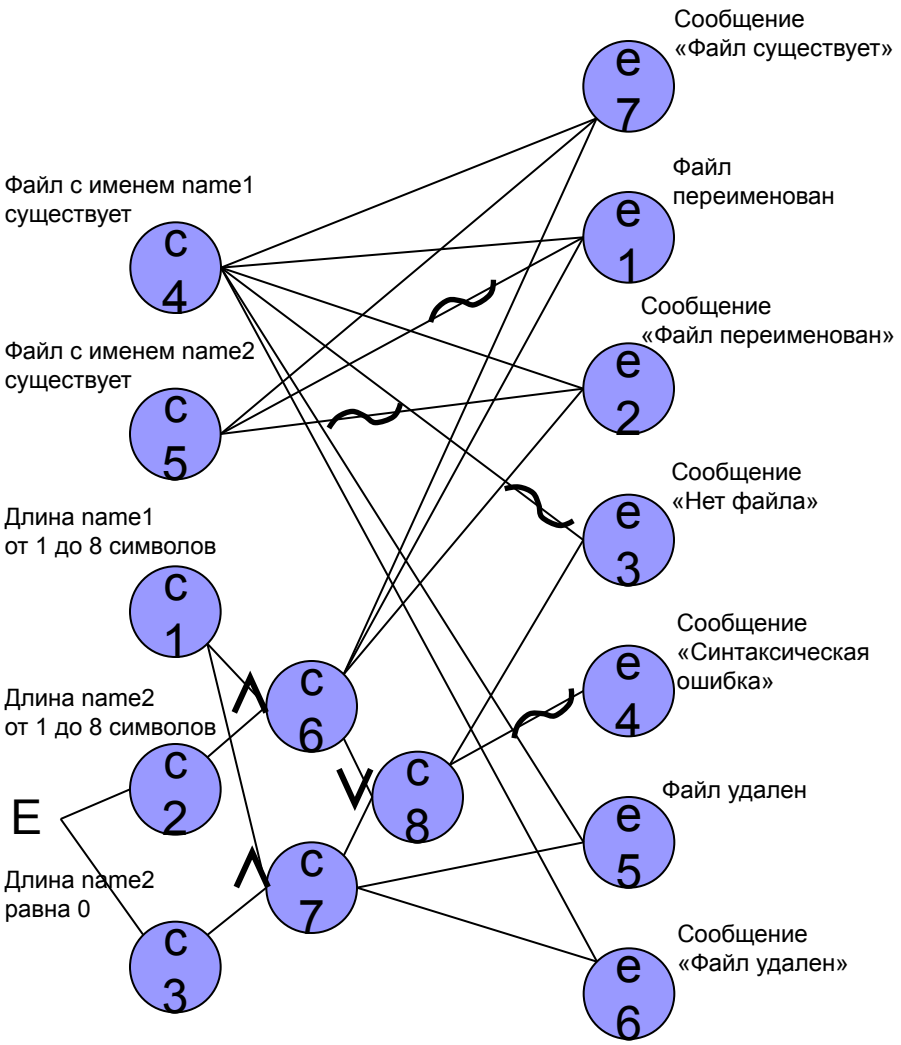


Таблица решений

Причины	Тесты						
	1	*2*	*3*	*4*	*5*	*6*	*7*
c1	1	1	1	0	0	1	1
c2	1	1	0	1	0	0	1
c3	0	0	1	0	1	1	0
c4	1	0	0	X	X	1	1
c5	0	X	X	X	X	X	1
Следствия							
e1	1	0	0	0	0	0	0
e2	1	0	0	0	0	0	0
e3	0	1	1	0	0	0	0
e4	0	0	0	1	1	0	0
e5	0	0	0	0	0	1	0
e6	0	0	0	0	0	1	0
e7	0	0	0	0	0	0	1
Рассматри-ваемое следствие	**e1** *e2*	**e3** “или” перед c8		**e4** “или” перед c8		**e5** *e6*	**e7**

Рассматриваются только те следствия, которые не были рассмотрены ранее