Обеспечение качества модификации сложных программных систем высокой надежности[1]

В.Г. Промыслов, Е.Ф. Жарко, О.А. Промыслова (ИПУ РАН)

АСУТП в последнее десятилетие вышли на качественно новый уровень развития, и одним из следствий данного процесса является существенное усложнение ПО, включаемого в АСУТП, и его жизненного цикла. В ИПУ РАН в 2000 г. был разработан комплекс системного ПО LICS для перспективных АСУТП АЭС. Требуемый срок службы всего АСУТП АЭС составляет не менее 30 лет, что приводит к необходимости разработки и внедрения значительных по объему и сложности процедур по сопровождению и модернизации программных комплексов. В работе рассмотрен опыт, накопленный ИПУ РАН в процессе сопровождения системного ПО LICS по обеспечению качества модифицированной программы и прогнозирование затрат на ее модернизацию и сопровождение.

Введение

АСУТП в последнее десятилетие вышли на качественно новый уровень развития, связанный с возросшим уровнем автоматизации объектов управления и, как следствие, ростом числа диагностических и управляющих сигналов, обрабатываемых системой в единицу времени. С другой стороны, практически линейный рост производительности вычислительных систем, которые могут быть использованы в АСУТП, позволил реализовать значительно более сложные алгоритмы управления и анализа данных с использованием сложных ПТК. Однако произошедший качественный скачок в составе решаемых задач заставил пересмотреть соотношение составляющих жизненного цикла программ.

Данные изменения хорошо прослеживаются на примере разработки ПО для АСУТП АЭС с требуемым сроком службы всего АСУТП АЭС не менее 30 лет. Это значительно превышает средний достигнутый на данный момент срок службы и хранения технических средств и заставляет уделять больше внимания тщательной разработке этапа модификации и сопровождения разработанного ПО.

Тенденция к перераспределению ресурсов, выделяемых на различные этапы жизненного цикла ПО, находит отражение и в нормативных документах, используемых для его разработки. Если стандарт СССР ЕСПД от 1977 г., до сих пор используемый программистами в России, еще не определяет и не использует понятие модификации видимо в виду его несущественности и (или) тождественности процессу разработки, то более поздние стандарты уделяют ему пристальное внимание [1].

Рассмотрим опыт, накопленный в течение более четырех лет модификации и сопровождения сложного высоконадежного системного ПО (СПО) LICS [2], разработанного в Институте проблем управления им. В.А. Трапезникова РАН для системы верхнего блочного уровня (СВБУ) АСУТП вновь строящихся и модернизируемых АЭС.

Сформулируем основные понятия и определения.

СПО – ПО, разработанное для специфической вычислительной системы или семейства систем и предназначенное для облегчения эксплуатации и обслуживания системы и соответствующих программ (например, ОС, компиляторы, утилиты). СПО обычно состоит из ОС и вспомогательных программных средств.

Модификация (модернизация) – внесение изменений в уже согласованные документы (программу).

Сопровождение – модификация программного изделия, связанная с исправлением ошибок, улучшением функциональности или производительности программного продукта или изменением окружения, в котором происходит выполнение программы.

Сопровождение и соответственно модификация ПО может быть нескольких типов: коррекция (тип А) – работы, связанные с необходимостью исправления ошибок в ранее разработанной программе; адаптация (тип В) – работы, связанные с изменением условий окружения, в котором выполняется программа; улучшение (тип С) – работы, связанные с добавлением новой функциональности в программу.

Для разработки и сопровождения СПО LICS использовалась "стандартная" модель жизненного цикла ПО [2]. Этапы жизненного цикла, связанные с разработкой СПО LICS изложены в [3]. Ниже будут подробно рассмотрены аспекты, связанные с этапом сопровождения и модификации высоконадежного ПО, обсуждены методы прогнозирования затрат на его сопровождение.

Сопровождение ПО

СПО LICS является сложным ПО с объемом кода около 200 Мб, включающим более 50 программных компонентов. Процедура его модификации и сопровождения должна гарантировать сохранение высокого качества предоставляемой функциональности и совместимости с прикладным ПО после проведения модификации.

[1]Работа выполнена при поддержке Российского фонда фундаментальных исследований грант № 05-0833372.

Этап жизненного цикла, связанный с сопровождением ПО, с процедурной точки зрения мало отличается от разработки нового ПО, являясь в определенном качестве рекурсивной процедурой в рамках жизненного цикла и может быть разделен на следующие фазы: 1) идентификация проблемы; 2) анализ проблемы (техническое задание); 3) технический проект; 4) разработка; 5) тестирование; 6) поставка (модифицированного ПО).

Фазы 3, 4 и 6 по своей сути тождественны соответствующим фазам в разработке нового ПО, при необходимости оценки качества модифицированного ПО используют методы, применяемые для оценки качества вновь разработанного ПО [3, 4]. Далее подробно рассмотрим состав работ, проводимых на фазах 1, 2 и 5.

Идентификация проблемы

Действия, проводимые на фазе идентификации проблемы этапа сопровождения подобны соответствующим работам, проводимым для нового ПО. И в том, и в другом случае основными исходными данными является наличие некоторой проблемы, которую намереваются устранить в ходе выполнения работ. Однако этап сопровождения обычно характеризуется лучшей специфицируемостью проблемы в результате наличия как у разработчика ПО, так и пользователя опыта работы в данной предметной области.

Фазе идентификации следует предварять деятельность, связанную с учетом несоответствий в функционировании ПО (термин несоответствие надо понимать широко и включить в него все типы сопровождения A-C). Данная деятельность является одной из ключевых для достижения высокого качества программного изделия [5]. Замечания по функционированию ПО оформляются в виде карточки учета замечаний. В оформленный документ с указанием замечания включены характеристические признаки дефекта: локализация проблемы, автор кода программы, содержащего ошибку, время, требуемое на внесение корректировки в код программы, чем было вызвано появление данной проблемы и ее обнаружение.

Результатом выполнения фазы идентификации должен стать выпуск запроса на модификацию с перечнем замечаний, которые предлагается устранить при модификации ПО. Выпуск запроса на модификацию имеет также цель аккумулировать и группировать выявленные несоответствия по признакам для облегчения проведения следующей фазы сопровождения ПО. В качестве системы классификации использована так называемая "Схема классификации по атрибуту" [6]. Согласно этой схеме данные по карточкам учета замечаний классифицировались по следующим атрибутам:

- категория (управление данными, определение исходных данных, системное ядро ОС);
- тип, поясняющий категорию (например, определение типа данных);
- наличие кода (несоответствие вызвано наличием лишнего кода, отсутствием кода, ошибкой в кодировании).

Введение данной классификации позволило точнее специфицировать причину и ситуацию появления несоответствия.

Анализ проблемы

Фаза анализа проблемы выполняется на основе запроса на модификацию и осуществляет оценку возможности и необходимости устранения каждого замечания, перечисленного в запросе. На основе анализа запроса на модификацию выпускается техническое задание с перечнем требований, которые необходимо удовлетворить в ходе проведения модификации ПО.

Наибольшую трудность для разработчика во время реализации фазы анализа часто составляет оценка не только технической реализуемости, но и трудоемкости проведения модификации. Кроме внесения изменений, связанных с реализуемой (новой) функциональностью, необходимо учитывать затраты на проверку влияния внесенных изменений на смежные области модифицируемого ПО и уже реализованной в оригинальном (немодифицированном) ПО функциональности. При оценке затрат на модификацию необходимо учитывать следующие факторы:

- степень модульности (независимости компонентов) модифицируемого ПО. Под модулем понимается программная единица, выполняющая некоторую независимую функцию в составе ПО. Чем более обособленно распределена по группам модулей модифицируемая функциональность, тем меньше затрат на тестирование смежных модулей;
- объем кода в каждом из модулей может существенно отличаться друг от друга (рис 1), и, следовательно, сложность модификации зависит от того, насколько "сложные" модули вовлечены в процесс модификации. Если модификация вносится в код, обладающий меньшей сложностью, то легче проведение анализа модифицированного модуля, меньше и затраты на тестирование реализованной функциональности.

Для оценки сложности ПО разработано большое число метрик, но их применение в большей мере отражает субъективный выбор эксперта, чем недостатки или достоинства данной метрики.

Во многих случаях для оценки сложности модуля (или группы модулей) может быть использована тривиальная метрика, основанная на подсчете суммы линий кода (ЛК). Однако объем кода программы по метрике ЛК не является стабильной характеристикой, отражающей ее сложность [7]. Авторами были использованы также две другие метрики: цикломатическая сложность (ЦС) [8] и метрика затраченных усилий (ЗУ) [9]. Для вычисления последней метрики не требуется проведения глубокого структурного анализа кода и она позволяет оценить число ошибок в коде и затраты, требуемые на его модификацию. Поэтому метрика ЗУ была выбрана основной, хотя для компактных модулей более приемлема оценка сложности с использованием метрики ЦС. На рис. 1 приведен объем кода по модулям, рассчитанный с ис-

пользованием метрик ЛК, ЗУ и ЦС и нормированный к максимальному значению. Нормированный объем кода, рассчитанный по всем трем метрикам, для большинства модулей близок, существенное различие объема по метрике ЦС для модуля № 30 вызван тем, что данный модуль является автоматически генерируемым и нетипичен для программного кода, созданного программистом.

Использование авторегрессионных моделей для анализа проблемы

Кроме качественных подходов к оценке затрат на модификацию ПО возможно и получение количественных оценок затрат на основании ранее накопленных данных о соотношении "сложность модификации-затраты на ее проведение".

Если возможно оценить сложность модификации, то легко перейти от нее к оценке затрат на проведение модификации. Чаще всего модификация не относится к какому-либо одному из перечисленных типов сопровождения A-C, а является композитной, то есть итоговая сложность модификации (обозначим ее Co_1) является некоторой функцией F от сложностей (Co) модификаций каждого из типов сопровождения:

$$Co_1 = F(Co(A), Co(B), Co(C)). \quad (1)$$

Далее следует сделать некоторые предположения о характере функции F и перейти к анализу возможности оценки $Co_1(X)$, $X = A,B,C$. При этом используем предположение о линейности функции F, хотя это предположение вызвано, прежде всего, желанием облегчить математические выкладки. Перепишем (1) в виде:

$$Co_1 = A_1\, Co(A) + A_2 Co(B) + A_3\, Co(C) + A_4. \quad (2)$$

Не теряя общности, положим в уравнении (2), что $A_1 = A_2 = A_3 = 1$, $A_4 = 0$. Теперь сделаем дальнейшие предположения о характере и вкладе сложностей типов A и C в итоговую сложность Co_1. Легко видеть, что вклад составляющих $Co(A)$ и $Co(B)$ при проведении модификации $n = N$ (n>1) в отличие от $Co(C)$, зависит от предыстории ПО, то есть от предыдущих модификаций и, в конечном счете, от сложности ПО на модификациях $i < n$. Составляющая $Co(C)$ является независимой от текущей сложности ПО. Модель временного ряда, использованная для оценки сложности модификации версии n (обозначим ее $Co_1[n]$) по предыдущим значениям $Co_1[i]$, $i = 1,...,n$-1 записана линейным разностным уравнением, известным как уравнение авторегрессии:

$$Co_1[n] = -\sum_{i=1}^{P} Co_1[n-1] \times B[i] + u[n], \quad (3)$$

где первый член уравнения (3) описывает вклад в сложность модификации составляющих сопровождения типа A и B, а $u[n]$ описывает вклад типа C, P – порядок авторегрессии. Для расчета необходимых параметров авторегрессии $B[i]$ нами использован алгоритм Берга [10]. При выборе порядка авторегрессии P в уравнении (3) необходимо учитывать характеристику модифицируемого ПО, при этом более надежному ПО

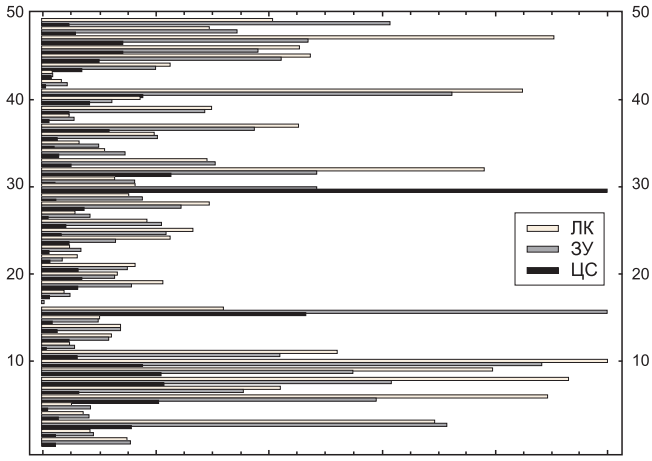

Рис. 1. Объем 49 модулей СПО LICS по метрикам ЛК, ЗУ и ЦС

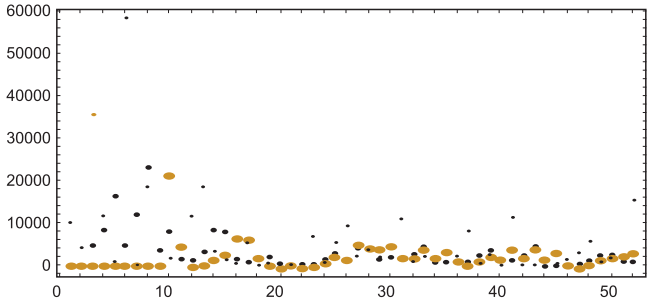

Рис. 2. Результат использование авторегрессионных моделей: P = 2 (точки черного цвета) P = 9 (точки рыжего цвета) для предсказания сложности модификации. Для сравнения показана реальная сложность модификации по ЗУ (черные точки маленького размера)

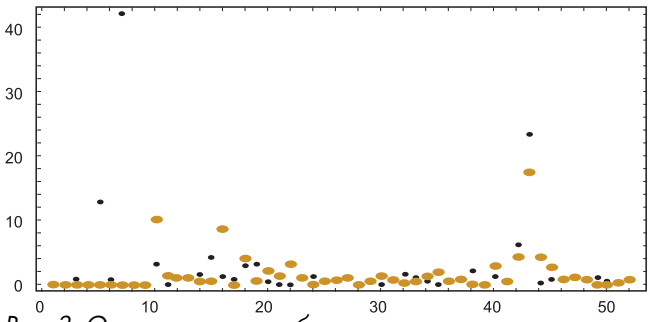

Рис. 3. Относительная ошибка предсказания для авторегрессионных моделей размера P = 2 (точки черного цвета) и P = 9 (точки рыжего цвета) для сложности модификации по ЗУ

соответствует большее значение параметра P. На порядок авторегрессии также влияет преемственность методов программирования (стиль, язык, используемые технологии) на каждом из циклов модификации, поэтому следует стремиться к разумному консерватизму и избегать без необходимости изменения стиля. Более консервативный подход обеспечит возможность использования авторегрессионных моделей большего порядка и уменьшит ошибку предсказания.

Для сравнения сделана оценка $Co_1[n]$, $n = P...52$ для $P = 2$ и $P = 9$. На рис. 2 приведены результаты оценок сложности модификации $Co_i[n]$, и на рис. 3 –

модуль относительной ошибки предсказания вперед, $|Co_1[n] - Co'_1[n]|/Co'_1[n]$, где $Co'_1[n]$ – экспериментальное значение сложности модификации. Для расчета была использована метрика ЗУ, хотя нет препятствий использовать и метрики других типов. Величина ошибки в оценке Co_1 определяется прежде всего вкладом члена $u[n]$, то есть модификации типа C. На рис. 3 видно, что в большинстве случаев относительная ошибка предсказания не превосходит по модулю реального (экспериментально определяемого) значения сложности, это дает возможность использовать данный подход для оценок затрат на модификацию.

Тестирование

Тестирование как составная часть оценки качества ПО производится на протяжении всего жизненного цикла ПО. С одной стороны, любая программа благодаря тому, что реализует некий известный алгоритм работы может быть полностью проверена при ограничении входного набора данных. Однако это утверждение правомерно только для тривиальных либо некоторых специальных случаев ПО. Ресурсы, выделяемые на тестирование, в большинстве случаев недостаточны для обеспечения полного тестового покрытия программы. Возникает задача классификации частей программного кода для того, чтобы выделить участки, наиболее вероятно содержащие ошибки.

Анализ тестируемости кода СПО на этапе сопровождения для сложного программного комплекса включает как априорную (перед началом модификации), так и апостериорную (после изменения кода) оценки. В качестве априорных оценок возможно использовать показатели сложности программных модулей (чем выше показатели сложности, тем исходный код тяжелее для изучения, контроля и модификации). В качестве апостериорной оценки может выступать объем модифицированного (нового) кода $V(T)$, где T – текстовая разность (различие), включающая удаленный и добавленный код программы для каждого из модулей, рассчитанный по выбранной метрике. Метрика выбирается из метрик, используемых для расчета сложности модуля кода и позволяет оценить затраты, требуемые для тестирования новой функциональности.

Тестирование в процессе сопровождения обычно разбивается на модификационное и регрессионное [1]. Первое призвано подтвердить корректность вносимых в программу изменений, второе – сохранить ранее реализованную функциональность.

В процессе тестирования первого типа объем тестов выбирается из условий необходимости обеспечить тестовое покрытие не менее, чем в объеме измененного кода программы.

Для проведения тестирования второго типа встает вопрос, какой объем тестового покрытия считать достаточным для подтверждения неизменности ранее реализованной функциональности. Авторы использовали подход, основанный на предположении (обоснованном для сложных программ), что для выполнения каждой функции (функциональности) используется не весь код программы, а некоторая его часть объемом V_f.

Эксплуатацию программы можно охарактеризовать выполнением набора функции $F_i(u)$, $i = 1...K$, где $u \in R$ – входные параметры.

Если V_f не всегда технически возможно определить для каждой из заявленной функциональности $F_i(u)$ из-за существования общих частей кода для каждой из реализаций функциональности в программе, то логично оперировать усредненным значением $\overline{V}_f = (1/K) \int_F V(t)df$ по всем возможным входным аргументам. Тогда объем тестового покрытия V_t для регрессионного тестирования следует выбирать из условия $V_t < L < \overline{V}_f$, где $L \le K$ – число ранее реализованных функций в программе, затронутых модификацией.

Зависимость качества программы от объема модификации

Интуитивно понятно, что качество программы для r-той версии зависит от объема модификации, которая проведена в предыдущих версиях программы. На рис. 4 слева показаны нормированные к максимальному значению экспериментальные зависимости двух величин (Co и Ex) для СПО LICS, где Co – сложность модификации по метрике ЗУ, Ex – оценка изменения качества программы от номера версии программы, справа показана нормированная корреляционная функция Nc от Co и Ex.

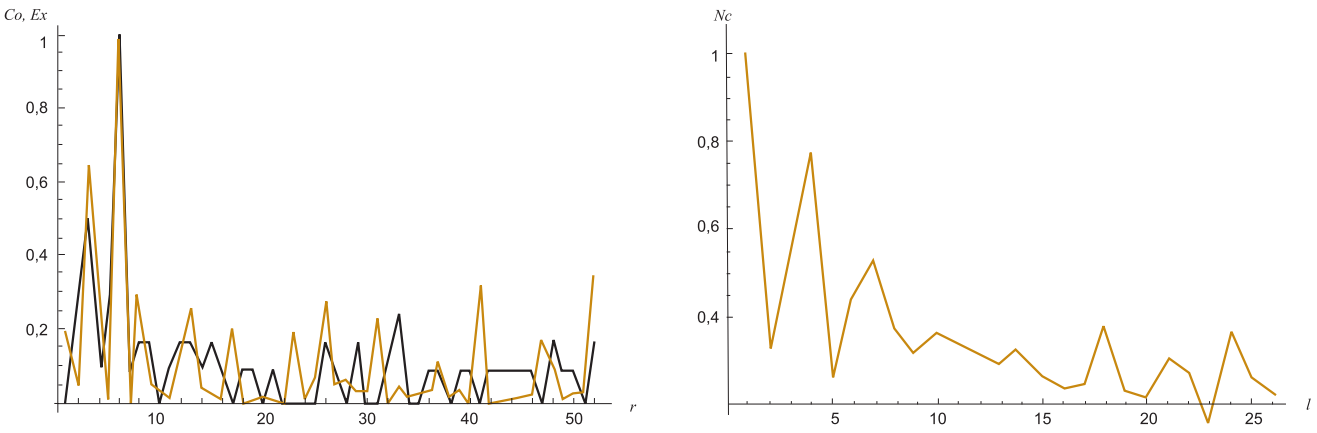

Рис. 4

Величина *Ex* оценивалась экспертным методом как число устраненных функциональных ошибок в программе во время модификации версии *r*. Можно заметить, что *Co* и *Ex* являются коррелированными, причем, если наибольшая корреляция достигается при нулевом сдвиге (это отражает очевидный факт, что большему объему изменений соответствует большее число исправленных ошибок), то существование второго значительного по величине пика корреляции рядом с первым характеризует зависимость качества версии r от объема изменений в версиях непосредственно предшествующих *r*, т.е. сам процесс модификации, связанный как с устранением ошибок, так и с внесением новой функциональности является источником ошибок.

Заключение

В результате работ по созданию СПО СВБУ АСУТП АЭС в ИПУ РАН на базе свободно распространяемой ОС Linux был создан коммерческий продукт СПО LICS. Достигнут высокий уровень качества ПО, разработана и испытана процедура сопровождения высоконадежного ПО, гарантирующая сохранение работоспособности и преемственности между версиями программы. Предложено применение авторегрессионных моделей для предварительной оценки сложности и затрат на модификацию ПО.

Список литературы

1. IEC 60880 Ed. 1.0 b:1986, Software for computers in the safety systems of nuclear power stations.
2. IEEE/EIA 12207.0 – 1996 Software life cycle processes
3. *Масолкин С.И., Промыслов В.Г., Жарко Е.Ф., Антонов А.В. и др.* СПО LICS как компонент подсистем АСУ ТП АЭС // Автоматизация в промышленности. 2004. №10.
4. *Липаев В.В.* Обеспечение качества программных средств. Методы и стандарты. М.: СИНТЕГ, 2001.
5. *Grady, Robert B.* (1996). Software Failure Analysis for High-Return Process Improvement Decisions. Hewlett-Packard Journal, 47(4) (August).
6. *Ostrand, Thomas S., and Elaine Weyuker* (1984). "Collecting and Categorizing Software Error Data in an Industrial Environment." The Journal of Systems and Software, 4:289-300.
7. *Haapanen Pentt, Pullkinen Urho.* Licensing process for safety-critical software-based systems. STUK-YTO-TR 171. Helsinki 2000.
8. *Halstead M.H., McCabe,* T. 1976. A Software Complexity Measure, IEEE Trans. Software Engineering 2; 12; (Dec 1976).
9. *Halstead M.H.* Elements of Software Science. New York: Elsevier, 1977.
10. *Марпл-мл. С.Л.* Цифровой спектральный анализ и его приложения. М.: Мир, 1990.

Промыслов Виталий Георгиевич *– канд. физ.-мат. наук, ст. научный сотрудник,*
Жарко Елена Филипповна *– канд. техн. наук, ст. научный сотрудник,*
Промыслоова Ольга Александровна *– научный сотрудник Института проблем управления им. В.А. Трапезникова РАН.*
Контактный телефон (495) 334-25-22. E-mail: vp@ipu.rssi.ru
Http://www31.ipu.rssi.ru/~lics

Новые многопортовые промышленные Ethernet-коммутаторы от MOXA

Компания "Ниеншанц-Автоматика" – стратегический партнер MOXA Technologies в России от лица производителя анонсирует выход двух новых моделей промышленных Ethernet-устройств, оснащенных большим числом портов. В июне появятся две новинки: неуправляемый коммутатор EDS-316 и управляемый EDS-518.

Модель EDS-316 имеет 16 портов 10/100 Мбит/с, из которых один или два могут быть оптоволоконными с поддержкой одно- или многомодового волокна. Модель EDS-518 также имеет 16 портов Fast Ethernet и еще оснащена двумя портами Gigabit Ethernet и интеллектуальными функциями управления сетью. Такие функции предназначены, прежде всего, для оптимизации пропускной способности сети в системах управления РВ. Чтобы передавать критичную информацию с минимальными задержками, в устройстве заложен ряд сервисов, например, назначение приоритета обслуживания, разделение групп пользователей и создание виртуальных подсетей. Стабильная работа системы, построенной на основе коммутаторов MOXA EDS-518, обеспечивается возможностью построения резервированных каналов связи по топологии MOXA Turbo Ring. Благодаря этой функции время автоматического восстановления соединения при обрыве Ethernet-сети не превышает 300 мс.

Жесткие промышленные условия эксплуатации предъявляют повышенные требования к надежности техники. Дополнительным фактором надежности коммутаторов является функция резервирования электропитания. В случае перебоев в электроснабжении или при обрыве Ethernet-связей коммутаторы замыкают встроенные контакты реле, к которым пользователь может подключить устройства сигнализации для оперативного получения информации о неполадках и локализации неисправности.

Http://www.moxa.ru

Siemens сделает автомобили безопасней

Специалисты "Сименс ФДО Аутомотив" разработали акустический датчик столкновения – CISS (Crash Impact Sound Sensor). В применяющихся сейчас системах сигнал о введении в действие подушек безопасности поступает от акселерометров (при лобовом столкновении) или датчиков давления (при боковом ударе). Датчики этого типа не в состоянии определить тяжесть аварии. Между тем, иногда значительно большую опасность для человека в автомобиле представляет "выстреливание" подушки безопасности, чем само столкновение.

Акустический датчик, фиксирующий вибрации, возникающие при деформации конструкции, различает "громкость" столкновения и при "тихой" угрозе способен сделать подушку менее жесткой. К тому же он действует быстрее нынешних датчиков, поскольку деформации конструкции предшествуют ударному замедлению. Новый датчик в настоящее время проходит испытания и должен быть готов к использованию в 2007 г.

Http://www.siemens.ru