The magazine of the ACCU

www.accu.org

{cvu}

Volume 21 Issue 1 March 2009 £3

Features

{cvu}

Volume 21 Issue 1
March 2009
ISSN 1354-3164
www.accu.org

Editor
Tim Penhey
cvu@accu.org

Guest Editor
Roger Orr
rogero@howzatt.demon.co.uk

Contributors
David Binkley, Ian Bruntlett, Pete Goodliffe, Paul Grenyer, Seweryn Habdank-Wojewódzki, Derek Jones, Dawn Lawrie, Dietrich Leichs, Alan Lenton, Anna-Jayne Metcalfe, Seb Rose, Matthew Wilson

ACCU Chair
Jez Higgins
chair@accu.org

ACCU Secretary
Alan Bellingham
secretary@accu.org

ACCU Membership
Mick Brooks
accumembership@accu.org

ACCU Treasurer
Stewart Brodie
treasurer@accu.org

Advertising
Seb Rose
ads@accu.org

Cover Art
Pete Goodliffe

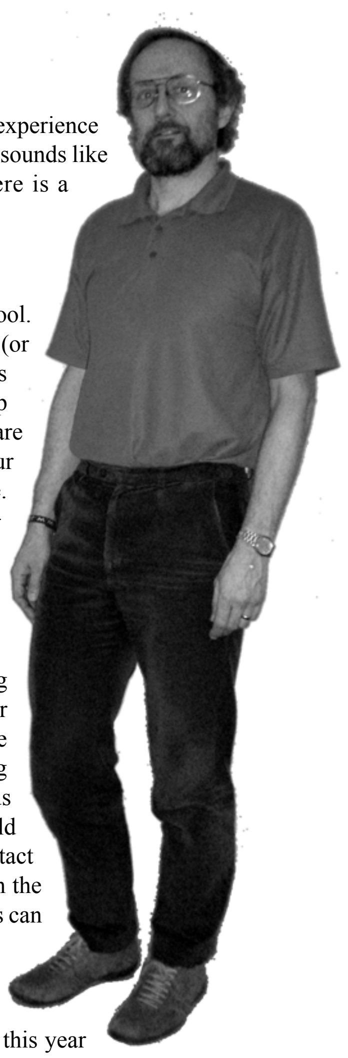
Repro/Print
Parchment (Oxford) Ltd

Distribution
Able Types (Oxford) Ltd

Design
Pete Goodliffe

accu

What do you want to know tomorrow?



I came across this sentence the other day on the BCS website: 'A depth of experience in one discipline and breadth of coverage across a range of others.' This sounds like a good principle for anyone in our industry, particularly when there is a downturn, but how do we get there?

Depth first

In my own experience many people are content to be a big fish in a small pool. If you have worked in the same area of IT for a while in the same business (or related ones) you may find yourself a 'local expert' in one discipline. In this position it is all too easy to stagnate and to stop learning. It's good to keep pushing ourselves out of our comfort zones – meeting other people who are also experts in the same discipline can help us to re-assess the extent of our knowledge. However this can be hard to achive in our normal working life. Through my membership of ACCU I have got to know many good C++ programmers and I've undoubtedly learned a great deal from them (as well as realising how much more there is to know!)

Breadth

Learning new things can be both fun and scary. In the early stages of using a new technology we don't know enough to even find answers to our questions; those we work with directly may be unable to help either if the area we are moving into is new to our company. We turn from being relatively comfortable experts to being unconfident novices – most of us don't enjoy this very much. It is often quicker to solve problems with the old technology we do know than with the new stuff. ACCU gives us ways to contact other people with the experience to point us in the right direction; through the mailing lists, regional groups, magazines and the conference. Such contacts can help us past those basic questions that are easy for the experienced person to answer.

Further opportunities to learn new skills include writing for CVu or Overload or speaking at a regional group or at the conference – although this year we had so many good proposals we sadly couldn't fit them all in. Or even guest editing CVu ... and on that note Steve Love will be editing the next CVu.

I look forward to meeting many of you in Oxford at the conference.

Roger

ROGER ORR
GUEST EDITOR

The official magazine of ACCU

ACCU is an organisation of programmers who care about professionalism in programming. That is, we care about writing good code, and about writing it in a good way. We are dedicated to raising the standard of programming.

ACCU exists for programmers at all levels of experience, from students and trainees to experienced developers. As well as publishing magazines, we run a respected annual developers' conference, and provide targeted mentored developer projects.

The articles in this magazine have all been written by programmers, for programmers – and have been contributed free of charge.

To find out more about ACCU's activities, or to join the organisation and subscribe to this magazine, go to www.accu.org.

Membership costs are very low as this is a non-profit organisation.

FEATURES

DIALOGUE

REGULARS

SUBMISSION DATES

C Vu 21.2:	1st April 2009
C Vu 21.3:	1st June 2009
Overload 91:	1st May 2009
Overload 92:	1st July 2009

WRITE FOR C VU

Both C Vu and Overload rely on articles submitted by you, the readers. We need articles at all levels of software development experience. What are you working on right now? Let us know!

Send articles to cvu@accu.org. The friendly magazine production team is on hand if you need help or have any queries.

ADVERTISE WITH US

The ACCU magazines represent an effective, targeted advertising channel. 80% of our readers make purchasing decisions or recommend products for their organisations.

To advertise in the pages of C Vu or Overload, contact the advertising officer at ads@accu.org.

Our advertising rates are very reasonable, and we offer advertising discounts for corporate members.

COPYRIGHTS AND TRADE MARKS

A Practical Introduction to the YAMI Library

Seweryn Habdank-Wojewódzki and Dietrich Leichs demonstrate inter-language object message passing.

This article describes some basic features offered by the YAMI library [1], which is designed to pass messages and their parameters through the network (using TCP/IP protocol). The acronym comes from 'Yet Another Messaging Infrastructure'.

This article is neither a tutorial nor a reference manual – these are available from the YAMI website – but instead concentrates on a practical example of the library usage. YAMI's interface is currently available in C, C++, TCL, Python, PHP and Java (demonstrated in four of these below). The general structure of each message contains the target, the name of the object, the name of the message and its parameters where the target can be either IP address or domain name. These features help to develop distributed and/or multi-language object communication. A draft of such a framework is shown in this article. (All the project code is available from the accu web site at http://accu.org/content/journals/cvu211/yami_example.zip)

YAMI as a communication library

There are many existing ways to implement communication in a distributed system. Some popular standards include: Common Object Request Broker Architecture (CORBA) [2], Internet Communications Engine (ICE) [3], Simple Object Access Protocol (SOAP) [4], Java Remote Method Invocation (RMI) [5], Distributed Component Object Model (DCOM) [6], Abstract Syntax Notation One (ASN.1) [7] and D-BUS [8]. Each of these has its own strengths and weaknesses but even a brief description of each of them is out of the scope of this article. So why would you choose YAMI – what are its features?

Some key features of YAMI

YAMI uses the XDR standard [9] which is a binary standard. This helps make YAMI fast – its performance on a Dual Core AMD 1800MHz Opteron can reach 25,000 messages per second. Another feature of YAMI is the simplicity of the usage and interface. Astronomers from Milan who were not skilled in programming comprehended it within 2 hours and were then able to write Python scripts for their distributed system. The simplicity of YAMI is increased by making the code using it as similar as possible in all supported languages. The core of understanding YAMI is to know the network agent; generally in Object Oriented (OO) languages (C++, Java, Python) the application code using YAMI is more or less the same. YAMI also supports P2P connections, which is useful to send messages through firewalls. YAMI does not require any kind of system 'daemon'.

Failover in YAMI is very simple: it just resends a message if it fails. In order to avoid an infinite repetition there is an internal mechanism which limits the duration of the retry. For simplicity in the presented example, error checking is not shown in full. However YAMI offers an error reporting functionality. In C the error reporting is supplied by an error code returned by the executed function, for OO languages exceptions are used.

What is not supported in YAMI 3.x?

YAMI is intended as a simple building block with few external dependencies. There is therefore no in-built support for distributed transactions – such a feature can be built on top of YAMI if required, for example using handshaking with message IDs. Similarly there is no implicit way to use encryption; in YAMI any encryption library used must be explicitly included and data have to be explicitly encrypted. The lack of encryption was a design decision to avoid the library relying on any encryption library. High level communication in distributed systems can be assisted by the use of name servers to bind service names to addresses but such searching is out of the level of YAMI's abstraction which uses IP addresses or network names like 'mydomain.com'. If a custom naming mechanism is required, this can be done manually. (Note that one of the tools related to YAMI is the Property Server which can assist in such an implementation).

To summarize, YAMI is not able to solve *all* the problems of distributed systems communication, and it does not even aim to do so. But YAMI is simple and easy to use – the next sections of the article will demonstrate this.

YAMI's features

Lightweight (library size of around 100kB or below, depending on the operating system).

Low memory and resource consumption, which is also fully tunable.

Load-balancing capability with message forwarding.

Automatic recovery from connectivity problems.

Comprehensive thread management options. It is possible to switch off threads, which is important for safety or mission critical systems.

Extremely easy and straightforward API for scripting languages.

Ability to 'bypass' firewalls with reverse message routing.

Portability across wide range of compilers, operating systems and hardware platforms.

Description of the example

Consider a building with restricted areas where a user must be granted privileges in order to enter certain rooms. At the entrance of every room there is a fingerprint sensor with a network interface. The aim of the system is to track people's movements – information about entry attempts by those without appropriate access may be particularly interesting.

The overall structure is shown in figure 1, where the numbers on the figure are the port numbers on the local machine. The first component is the `Sensor` that detects attempts to enter the room and records the recognized ID together with a time-stamp and the source system (both these pieces of information are important in real-time applications). The second element of the application is the `Buffer`. The information from the `Sensor`s is sent to the `Buffer`s – note many `Sensor`s can be connected to a single `Buffer`. All the `Buffer`s are in turn connected to the server (`Object Broker`) which will collect all data from the `Buffer`s and store them in

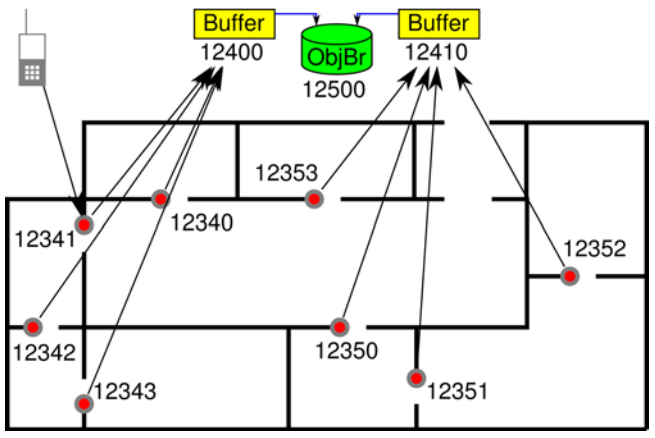
DIETRICH LEIHS

Dietrich is a long time experienced manager with a very strong background in electronics, signal processing and telecommunication. He sails in his free time. He can be contacted at dietrich@kruenes.cc

SEWERYN HABDANK-WOJEWÓDZKI

Seweryn specializes in high-performance distributed computing. He is also focused on the industrial quality standards of the code. In his leisure time he enjoys self-made art. Seweryn can be contacted at habdank@gmail.com.

Figure 1



a database; in this simple example the storage format is a CSV file (Comma Separated Values). The last element of the system is a **`Terminal`**. This program is designed to work directly with the sensor; sensors usually contain a list of the privileged persons and the purpose of the terminal is to operate on the list. Terminals can be run on mobile phones, which enables an operator to tune the parameters of the sensor on the spot.

Messages used in the example.

The **`Terminal`** sends **`"del_id"`** and **`"add_id"`** messages to the **`Sensor`**. The **`Sensor`** sends to the **`Buffer`** all new data by using the **`"new"`** message. The **`Object Broker`** requests data from the **`Buffer`** by using **`"get"`** message. Both **`Buffer`**s and **`Sensor`**s react to the **`"shutdown"`** message.

The fingerprint sensor is written in pure C and uses the implementation of YAMI written in C. The buffers are written in C++ using the YAMI C++ wrapper. The Data server (Object Broker) is written in Python. Normally such applications are set up on fast machines, so the Python script can be efficient enough. The terminal is written in Java. The presented source code is simple, it contains all the functionality that is important but omits other details such as network security.

Fingerprint sensor

Listing 1 contains key lines from the C code for the fingerprint sensor (`fp_sensor.c`). The sensor consists of two main elements. The first one is the client which gets the user IDs and sends them to the buffer. The second one is the internal server which processes messages for changing the list stored in the sensor.

While the simulator is running it sends data (sensor ID, time stamp and human ID) to the buffer every second (line 153). The *server* is the most important part of the sensor and contains the code to allow changing of the Sensor parameters. First the FpSensor object is registered with the agent (line 114). The agent listens for messages which generate events when the function servant is executed (line 251). In this function the handler for the parameters which can be sent through the network with messages is prepared (line 253). The command gets the message name from an incoming message (line 264) and then fixed messages that are stored in the server are compared with the incoming ones (lines 271 and 285). The exit message has no parameters, so its functionality is very simple – all resources have to be released before exiting, but this program relies on the operating system to do that. The parameters are taken from the message by using the **`yamiAgentIncomingMsgGetParameters`** command (line 287). This command takes all parameter sets from the message. The **`yamiGetStringLength`** takes the length of the stored string. The other function **`yamiGetStringValue`** takes a string from the parameter set. The message **`"del_id"`** has only one parameter. The removal of an ID from the list is simulated. The **`"add_id"`** message has two parameters: the first is the human ID and the second is the simulated string of the data that represents the internal structure of the fingerprint in the sensor memory.

Listing 1

```
063 int main (int argc, char * argv[]) {
...
077    HPARAMSET params;
...
080    HYAMIAGENT agent;
...
108    yamiNetInitialize();
...
114    yamiCreateAgent (&agent,
          pr_opts.fp_sensor_port, NULL);
...
126    while(1) {
...
128      yamiCreateParamSet (&params, 3);
...
136      yamiSetInt (params, 0, process_id);
137      yamiSetInt (params, 1, time_info);
138      yamiSetString (params, 2, human_id);
...
147      yamiAgentMsgSendAddr (agent,
            pr_opts.buffer_host,
148         pr_opts.buffer_port, 2,
149         "HID", "new", params, NULL);
...
153      sleep(1);
154   }
...
157    yamiDestroyAgent (agent);
158    yamiDestroySemaphore (sem);
159    yamiNetCleanup();
...
164 }
...
251 void servant (HINCMSG incoming) {
...
253    HPARAMSET params;
...
264    yamiAgentIncomingMsgGetMsgName (incoming,
          &msgname);
...
271   if (strcmp (msgname,
         FpSensorMsg_DELID) == 0) {
...
283   }
284
285    if (strcmp (msgname,
          FpSensorMsg_ADDID) == 0) {
286      /* get parameters from message */
287      yamiAgentIncomingMsgGetParameters(
288         incoming, &params);
..
293      yamiGetStringValue (params, 0, id);
...
303   }
304 }
```

The code of the client is placed in the **`main`** function. The handler for the parameter set is defined (line 77) and the client is constructed (line 80), defined as an agent. The agent is responsible for all the communications with the network environment. This environment is initialized (line 108) and then the agent is created (line 114). Finally, at the end of **`main`**, there is code to clean up the agent and the network on loop termination (lines 157–159).

The **`while`** loop contains all the client functionality (lines 126–154). What is the client doing? The client creates a parameter set with the length of 3 elements (line 128) – it is important to create a structure with the right size at the start. Next the actual information like sensor ID, time stamp and human ID is prepared and then put in the set (lines 136–138) respecting

Listing 2

```
036 namespace human_ids {
...
046    class Buffer {
047    public:
...
052     ::std::auto_ptr <Buf_t> get_human_ids() {
...
054       ::YAMI::MutexHolder mh (mtx);
...
059     }
...
062      void put_human_id (::std::auto_ptr
            < ::YAMI::ParamSet > & human_id) {
...
064       ::YAMI::MutexHolder mh (mtx);
...
066     }
...
071   };
...
074    template <typename Buffer>
075    class Server :
          public ::YAMI::PassiveObject {
076    public:
...
086      void call (::YAMI::IncomingMsg
            & incoming) {
087        String msgname (
              incoming.getMsgName());
...
090        if (msgname == msg::GET) {
...
097          Buf_t_ptr buf (get_hidbuf_());
...
102          ::YAMI::ParamSet params (
               bs * NO_HID_PARAMS);
...
116          incoming.reply (params);
...
118       }
...
121        if (msgname == msg::NEW) {
...
123          ::std::auto_ptr < ::YAMI::ParamSet >
124             params (incoming.getParameters());
125          if (params.get() != NULL) {
126             put_human_id_ (params);
127          }
...
132       }
```

Listing 2 (cont'd)

```
...
134        incoming.eat();
...
136     }
...
146   };
...
150 } // namespace human_ids
...
152 namespace buffer {
...
161    class Server :
          public ::YAMI::PassiveObject {
162    public:
...
166      void call (
            ::YAMI::IncomingMsg & incoming) {
...
177        incoming.eat();
178     }
...
183 } // namespace buffer
184
185 int main (int argc, char * argv[]) {
...
194   ::YAMI::netInitialize();
...
197     ::YAMI::Semaphore sem (0);
...
199      buffer::Server buffer_server (sem);
200      human_ids::Buffer human_idsbuf;
...
203      human_ids::Server <human_ids::Buffer>
204        human_ids_server (::boost::bind(
              &human_ids::Buffer::get_human_ids,
205           &human_idsbuf),
206               ::boost::bind(
                &human_ids::Buffer::put_human_id,
207                       &human_idsbuf, _1));
...
210     ::YAMI::Agent agent (port);
...
221      sem.acquire();
...
229 }
```

both the types and values of the data. The integer value **`process_id`** emulates the real sensor ID, **`time_info`** is the time stamp from the begin of the system epoch, human ID (**`human_id`**) is a string. The **`yamiAgentMsgSendAddr`** function is used to send the message with parameters to the proper object (line 147); this function takes the handler to the agent, the address of the target, the port of the target and the communication level. (The level is not used in this example but it can be important for some systems, for example if only strings are passed.) The actual object sent is named HID, with message name **`"new"`** and then its parameters. This single command sends the whole message (and also consumes the parameter set).

Note that there is no need for parsing and/or casting objects from one type to another – values with a concrete type are sent through the network. This provides at least two advantages: firstly, the performance increases due to the fact that there is no need of e.g. an XML parser; secondly, it is easy to prepare a serialization procedure for storing an object into the **`ParamSet`** (HPARAMSET) structure.

Buffer

The **`Buffer`** contains two logical objects. The first is the main buffer program itself and the second is the container for the ID numbers and the rest of the data. The code is shown in Listing 2 (`buffer.cpp`). Both objects are separated because the buffer reacts to messages like **`"shutdown"`**, and the HID object (container) reacts to the incoming data from the sensor and requests from the Object Broker. The C++ code contains two namespaces **`human_ids`** and **`buffer`**. Both contain **`Server`** classes. Every **`Server`** class is derived from **`YAMI::PassiveObject`** – this kind of server is passive, it means that it waits passively for the incoming messages. Both servers contain an overloaded **`call`** method which is called from the YAMI agent when a message arrives.

In detail, the network is initialized in **`main`** as in the C code (line 194). Next a semaphore is created (line 197) which blocks the buffer server (see line 221) and forces it to wait for all messages. Then the **`Buffer`** server functionality is initialized and the buffer for the human identification numbers is created (lines 199–200). Now the functions are bound to the server using **`boost::function`** function callbacks (line 203). The agent is created and joined with the servers by registering objects (lines 203–210).

Listing 3

```
...
025 class Buffer (object):
...
031 class HumanInfo (object):
032    def __init__ (self, fp_sensor_id,
           time_stamp,human_id):
...
046 def parse_config_file (config_file):
...
077 def main(argv):
...
100    agent = YAMI.Agent()
...
106        agent.domainRegister(str(i),
              str(domain_params[0]),
107           int(domain_params[1]), 2)
...
118             msg = agent.send(str(i),
119                buf.OBJECT, buf.GETMSG)
...
122             msg.wait()
...
125             status = msg.getStatus()
...
128             if status == YAMI.eReplied:
129               retpar = msg.getResponse()
...
159 if __name__ == '__main__':
...
169    sys.exit (main (sys.argv))
```

The **call** function from the **buffer::Server** is very simple. It gets the message name from the incoming message structure. If the message is **"shutdown"** then the semaphore is released which allows the program to exit. It is important to treat *all* incoming messages. **eat** is used in this example, but there are other possibilities such as **reject**, **reply** or **forward**.

The call method in **human_ids::Server** (line 86) is more complicated. It distinguishes between the two messages **"new"** and **"get"**. The first message forces the server to create a new object in the buffer from incoming parameters (lines 123–126). The **"get"** message comes from the Object Broker. The server creates a temporary buffer, takes information about the length of the internal buffer and then creates a parameter set (lines 97–102). Finally the function replies to the message (line 116), sending a list of all data to the Object Broker (transported through the network).

It is important to highlight that messages are events, and the functions connected with them can be executed at the same time, hence the need for a mutex in the **get_human_ids** and **put_human_id** methods in the **human_ids::Buffer** class (lines 54 and 64). This mutex blocks the operation on the container. Note that the **YAMI::MutexHolder** is used for this purpose as it is designed using RAII (Resource Acquisition Is Initialization) to ensure the mutex is correctly released. Finally the buffer collects data from the fingerprint sensors and could send them to the Object Broker on demand.

Object broker

The Object Broker is designed to collect all data from the system and to store them in the simple CSV data base. The Object Broker is written in Python (see Listing 3, `obj_broker.py`). Every 5 seconds (line 148) requests are sent to all buffers to get data and store them into the file.

The python script first steps through all classes definitions (**Broker**, **Buffer**, **HumanInfo**) and functions (**parse_config_file**, **main**). At line 159 the module is checked for being **"__main__"**, meaning that the file is being executed as a main program. In the **main** function (line 77) the configuration file is read and parsed, which contains ports and

Listing 4

```
12 public class Terminal {
...
17    public static void main(String[] args) {
...
36        TerminalExit t_exit =
             new TerminalExit();
...
42        CommandParser command_parser
43         = new CommandParser (t_exit, c_func);
...
50        while (true) {
...
55            command_parser.parseLine(line);
...
59        }
...
63    }
64 }
```

addresses of the buffers in entries like: **12400 127.0.0.1**. An agent is created (line 100) but without specifying an explicit port, which leaves the choice of port to the system. Then every buffer registers for its own domain (or ID) which is used for simplifying the communication procedures as the ID can be used instead of using addresses and ports. After initialization the Object Broker starts to collect data in a **while** loop. For every domain, the message **"get"** is sent to the object **Buffer** (line 118) and then the **wait** method blocks the current thread and waits for the reply. Once the **wait** completes the status of the message is checked (line 125). There are nine possible statuses, three of which are checked. If the message status is **eReplied** then the parameters are retrieved from the message using the **getResponse** method (line 129) to create a **HumanInfo** object from the incoming message and store it in the file.

The aim of the Object Broker operation is to get data from all of the items listed in the configuration file Buffers and save them into the data base (file). Once collected the data in the Buffers are erased.

Terminal

The aim of the terminal is to operate on the fingerprint sensor's internal data structures. As mentioned above, the sensor can be a standalone piece of hardware with its dedicated software. The terminal can be launched, for example, on the mobile phone. Again security aspects are omitted, but they can (and should!) be implemented. The terminal is a simple console application written in Java. The program waits for a command typed by the user and sends it to the sensor. There are some helper functions in order to filter out and forward known commands, this is just a skeleton of a possible procedure. The terminal handles three commands: **"add_id" <ID> <data>**, **"del_id" <ID>** and **exit**. The **exit** command exits from the terminal. The other commands have parameters that are sent to the sensor. In the **Terminal.java** (Listing 4, `Terminal.java`, line 36) a wrapper class for the **System.exit** function is constructed and a command parser is created and used. The class **FpSensorFunctions** (see Listing 5) implements the **Executable** interface which contains the method **execute**. In this method a simple message is prepared using the method **YAMIEasyClient.sendOneWay**: this method takes the address, port, name of the object, name of the command (which is a message in the YAMI nomenclature) and its parameters (see lines 30–32 in Listing 5).

Some results

For the purpose of testing the whole system is run in the structure shown in Figure 1. The example of the results from the sensor output stream is listed below (omitting the human readable time stamp for clarity).

```
Fingerprint sensor 12060 on port 12340 starts
communicate with buffer 127.0.0.1:12400
...
```

```
Fingerprint sensor 12060 sends human ID 56-72-38 at
5789755
...
Add ID 12-34-56 with data Data_buffer_example.
Fingerprint sensor 12060 sends human ID 81-70-30 at
5789782
...
Delete ID 12-34-56.
Fingerprint sensor 12060 sends human ID 80-77-72 at
5789789
...
Add ID 34-56-78 with data Another_data_buffer.
Fingerprint sensor 12060 sends human ID 42-72-21 at
5789803
```

The output from the buffer is:

```
  Buffer server started.
  'new' received.
  ...
  'get' received.
  ...
  'shutdown' received.
```

The output and input from the terminal is:

```
<OUTPUT> Terminal started for fingerprint sensor:
127.0.0.1:12340
< INPUT> add_id 12-34-56 Data_buffer_example
< INPUT> del_id 12-34-56
< INPUT> add_id 34-56-78 Another_data_buffer
< INPUT> exit
<OUTPUT> Terminal exits.
```

Messages sent from the terminal to the sensor can be observed (in the sensor results they are highlighted by using bold font).

The output from the object broker is:

```
  Domain 127.0.0.1:12400 added
  Domain 127.0.0.1:12410 added
  Object broker started.
  Get human ID
  ...
  Get human ID
  ...
  Exit.
```

And finally the result stored in the `data.csv` file:

```
  12060, 5789756, 50-55-14
  ...
  12064, 5789759, 65-18-84
  12060, 5789760, 30-21-33
```

Sensor IDs, timestamps and human IDs are stored. Note the sensor `12060` entries that are shown in the file. ■

Listing 5

```
public class FpSensorFunctions implements Executable {
...
   public int execute (String [] command_args) {
...
       if (command_args[0].equals(FpSensorTxt.DELID)) {
...
YAMIEasyClient.sendOneWay(fp_sensor_address_, fp_sensor_port_,
              FpSensorTxt.OBJECT, command_args[0],
              new ParamSet().append(id));
       }
...
       if (command_args[0].equals(FpSensorTxt.ADDID)) {
...
YAMIEasyClient.sendOneWay(fp_sensor_address_, fp_sensor_port_,
              FpSensorTxt.OBJECT, command_args[0],
              new ParamSet().append(id).append(data));
      }
...
   }
...
}
```

Acknowledgements

We would like to thank Maciej Sobczak for continuous help with the library usage. We also acknowledge Filip Wasilewski and Robert Wojciechowicz for remarks on the Python internals. We also thank Roger Orr for important remarks on the content and style for the article.

References

[1] http://www.msobczak.com/prog/yami/index.html
[2] http://www.omg.org/
[3] http://www.zeroc.com/index.html
[4] http://www.w3.org/TR/soap/
[5] http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp
[6] http://msdn2.microsoft.com/en-us/library/ms809311.aspx
[7] http://en.wikipedia.org/wiki/Abstract_Syntax_Notation_One
[8] http://www.freedesktop.org/wiki/Software/dbus
[9] http://www.faqs.org/rfcs/rfc1832.html and http://www.faqs.org/rfcs/rfc1014.html

Bluffer's Guide to the Semantic Web

Seb Rose tries to explain the meaning of semantic.

In the beginning, time out of mind, was the internet. We could send e-mails, post to bulletin boards, download files. Later, still shrouded in the mists of time, Tim Berners Lee invented the world wide web. Now we could surf for information, products and services, create personal content and interact globally. Somewhere on the way the phrase 'Web 2.0' entered common parlance, describing 'the changing trends in the use of World Wide Web technology and web design' [1], though many (including Mr. Berners Lee) believe that it doesn't amount to much.

And so to the Semantic Web. Is it new? When will it arrive? What's it for? How does it relate to Web 3.0? All good questions.

The term semantic web first got a public airing in 2001 in an article by Tim Berners Lee, Jim Hendler and Ora Lassila in the *Scientific American* [2]. This is not an academic paper, but a generally accessible description of what might be achievable on the web. The idea of adding semantics (meaning) to the web (which is generally about presentation) using Resource Definition Format (RDF) and ontologies (schema) is discussed, but not how this process could be facilitated. Now, 8 years later, there's not a lot to show the general surfing public that there has been any progress made in this direction. The most popular search engine is based on pattern matching textual query terms, and any attempt to combine information from disparate web sites is a depressingly manual (and frustrating) affair.

This is a very quick tour of the various semantic web technologies, so I'll be avoiding detailed explanations of grammars of syntax. I'll introduce the main W3C standards that apply, but you should remember that some of them are really quite recent. This may well be one reason why there has been so little obvious progress over the past 8 years. The web first came into existence largely on the back of HTTP, but there wasn't an equivalent technology for the semantic web to piggyback.

The Resource Description Framework (RDF) was a W3C recommendation as far back as 1999 and was an extension of entity-modelling that made statements about (web) resources. The W3C has this to say about RDF: 'RDF is a directed, labeled graph data format for representing information in the Web'. The format of an RDF statement (or assertion) follows the subject-predicate-object form, also known as triples (so called because it has three elements):

My car (subject) is powered by (predicate) methane (object).

Which can also be written as the triple: MyCar poweredBy Methane.

RDF comes with a number of predefined serialisations, some of which are simple renderings of triples (like the example shown above). There is also an RDF XML serialisation, which is in common usage. It should be remembered that RDF expresses set of triples, no matter which serialisation is used. RDF can now be embedded in web pages (using RDFa) to annotate displayed data with semantics. Since anyone can publish a web page, anyone can publish RDF data – you might not 'see' it because it is ignored by your browser's rendering engine.

RDF comes with a set of predefined resources, in its own namespace, so that we can build basic type systems. The other standards that layer upon RDF that are in common usage are Resource Description Framework Schema (RDFS) and the Web Ontology Languague (OWL). These also come with their own namespaces, stuffed full of more advanced predefined resources that allow us to build more expressive schemas. They allow us to declare classes that have transitive properties. They allow us to declare properties. And lots more, which allows us to build rich and complex type systems for data we publish on the web. Much of this ontology modelling will appear familiar to those of you who practise object orientated design, but beware.... there are subtle (and not so subtle) differences.

Now we can construct our own schemas, serialise them in RDF/XML and annotate our web pages. This in turn allows anyone to build software agents that are aware of the semantics of these standards to derive meaning from the data, rather than just being able to render it for us to look at. That's quite nice, but doesn't sound very compelling. Well, these semantics that these standards define come from the field of Description Logics (DL) and this is where things get interesting (and difficult). Using the semantics, we can construct general purpose inference engines that can take a set of triples and infer (deduce) other triples that also exist. Lets have a simple example:

Triple 1: Fred isSiblingTo Joe.

Triple 2: Joe isSiblingTo Anne.

Triple 3: isSiblingTo rdf:type owl:TransitiveProperty

Triple 4: isSiblingTo rdf:type owl:SymmetricProperty

Triple 1 and Triple 2 declare some facts: we know that Fred is Joe's sibling and that Joe is Anne's sibling. Triple 3 and Triple 4 tell us that the sibling relationship is symmetric (reflexive) and transitive. After applying an inference engine that understands RDF, RDFS and OWL we would end up with a virtual triple set that includes the triples:

Triple 5: Joe isSiblingTo Fred.

Triple 6: Anne isSiblingTo Joe.

Triple 7: Fred isSiblingTo Anne.

Triple 8: Anne isSiblingTo Fred.

This inference process is complex and processing intensive. And each layer of standards (RDF→RDFS→OWL) introduces more complexity and makes the job harder. If you are brave enough to look at some of the DL literature, you'll soon see that you can get into a state where some inferences are simply not decidable (in finite time). This has led to various subsets, supersets and species of these standards. Even in the OWL standard itself, 3 seperate species of OWL are defined – OWL Lite, OWL DL and OWL Full. Your choice will be governed by your requirements – if you need fast responses then you'll need to keep your schema simple; if you have a complex ontology and a large dataset, then expect to set your inference engine going and come back in a week.

Ignoring the performance issues, I hope you can see that there are some possibilities. If, for instance, all public transport companies annotated their online timetable systems using the same ontology (schema), then we could construct an intelligent software agent to plan a journey from A to B on a certain date, at a certain time, for a chosen price.

How likely is it that different companies would choose the same ontology? Well, there are quite a lot of general purpose ontologies out there, but we all know it is not likely that all systems will adopt the same one. The semantic standards come with a number of powerful facilities that let us map ontologies together. These facilities are just more RDF triples with predefined semantics, such as `owl:equivalentClass`. We then just run our inference engine over our tripleset to generate the equivalent triples and frame our queries using the our preferred ontology.

SEB ROSE

Seb is a software contractor in some of the most hostile working environments, including banks, pension providers and manufacturers. He works with whatever technology is prescribed by the client – mainly C++, .NET and Java. Recently he realised a lifelong ambition and became a tractor owner.

In the last paragraph I mentioned querying. Obviously with such a lot of data to process we'll want to query it, and this has been one of the areas where the semantic web has been (and still is to some extent) lacking. W3C published its recommendation for a semantic web query language, SPARQL, in January 2008. This is a recursive acronym apparently: SPARQL Protocol and RDF Query Language. There had been earlier query languages, RDQL and SERQL to name two, but W3C worked to provide a standard that satisfied diverse use cases. Not everyone is happy with the end result [3], but everyone never is. It is an RDF based query language that essentially matches conjunctions of triple patterns.

The recent W3C RDFa recommendation (mentioned earlier) provides a set of XHTML attributes to augment visual data with machine-readable hints. This allows sites to integrate RDF data into the rendered web page, rather than publishing it in seperate RDF documents. Now 'Anyone can say Anything about Any topic' (AAA), which of course means our software agents will need to decide which websites to trust. This is a recurring problem and there are no new solutions in the RDF world.

Is anyone using RDF et al?

- Some familiar sites like Facebook, Linkedin or Digg publish information in RDF.
- There are a number of standard ontologies that are in common use, such as Friend Of A Friend (FOAF).
- RDF was the basis of the Atom 1.1 standard, but that mainly got sidestepped by the more popular Atom 1.2.
- And, there are all the application listed in the Resources section below.

There is a lot of RDF data out there, but it tends to be in certain specialised domains.

The standardisation process has got to a state that there is now a useful set of technologies for application development. Over the coming years more and more data will be published as RDF thus making it more worthwhile to construct agents that can consume the data and do something useful with it. If the ongoing research into efficient execution of semantic queries over large datasets bears fruit, then we will really be able to reap the benefits of semantically tagged data.

Hopefully this has given a brief introduction to the major pieces of the semantic web jigsaw puzzle. I'll be covering these technologies in a lot more detail at my ACCU Conference session on Wednesday 22nd April at 2 o'clock.

Resources

The first book to read: *Semantic Web for the Working Ontologist* – Allemang/Hendler – ISBN 978-0-12-373556-0

Some open source/free tools and platforms that you can use to get you up and running:

- Protege – an IDE for developing ontologies. Read their tutorial on developing an ontology for pizzas.
- Neon – a European toolkit for ontology engineering.
- Sesame – an RDF database using the SERQL query language (and maybe SPARQL).
- Jena – an RDF database.
- Owlim – a semantic repository (you can use it as the storage layer for Sesame).
- Pellet - an OWL reasoner.

Some of the more interesting semantic web applications I've come across:

- DBPedia – a community effort to extract structured information from Wikipedia and to make this information available on the Web. DBpedia allows you to ask sophisticated queries against Wikipedia, and to link other data sets on the Web to Wikipedia data. (http://dbpedia.org)
- Sindice – Over 10 billion of application reusable pieces of information are on the Web today. They come from projects such as LOD but most of all from lots of Web 2.0 sites which are embracing the vision of facilitating easy sharing of information for subsequent automatic reuse and aggregation. (http://sindice.com)
- Watson – a semantic web search engine at: http://watson.kmi.open.ac.uk/WatsonWUI/
- Twine – a new way for you to collect online content – videos, photos, articles, Web pages, products – and bring it all together by topic, so you can have it in one place and share it with anyone you want. (http://www.twine.com)
- OpenCalais – The Calais Web Service automatically creates rich semantic metadata for the content you submit – in well under a second. Using natural language processing, machine learning and other methods, Calais analyzes your document and finds the entities within it. But, Calais goes well beyond classic entity identification and returns the facts and events hidden within your text as well. (http://www.opencalais.com)
- Freebase – an open database of the world's information. It's built by the community and for the community – free for anyone to query, contribute to, build applications on top of, or integrate into their websites. (http://www.freebase.com)
- paggr – winner of Semantic Web Challenge 2008. A novel online application that combines smart data with semantic agents and personalized portals. Not yet released. (http://paggr.com/about)
- iyouit – gathers data around you and about you. We call this data context. Context is centered on places you visit and people you meet and can grow to include all kinds of things that surround you. (http://www.iyouit.eu)
- xOperator – combines advantages of social network websites with instant messaging. (http://aksw.org/Projects/xOperator)
- OPO – aimed at enabling the integration and exchange of data related to a user's presence in the online world. As opposed to static user profiles and the interoperability in that domain, well supported by the FOAF Vocabulary, the Online Presence project targets the interoperability of dynamic properties that determine a user's current state of presence. (http://www.milanstankovic.org/opo/) ■

References

1. Wikipedia: http://en.wikipedia.org/wiki/Web_2.0
2. http://www.sciam.com/article.cfm?id=the-semantic-web&print=true
3. http://clarkparsia.com/files/pdf/sparqldl.pdf

Hunting the Snark

Alan Lenton has been searching for a job recently – it's not been as straightforward as he'd like.

Fun and games

The recruiter was succinct, and to the point. 'I've just got off the phone to the XYZ Sports Games studio', she said, 'They really liked the class design you prepared, but in view of the fact that you couldn't name an England football team for the next World Cup, they're not interested in considering you for the software developer job.'

This was something of a first. I've never been turned down for a job before because I couldn't name my own choice of an England football team. I couldn't even think of a book I could buy to help me over this massive gap in my technical ability.

As it happened, this was only the first in an increasingly surreal set of job hunting encounters.

The next day I received a phone call from another recruiter who wanted to put my CV in for a high level programming job with an online poker company. As it happens, I do have quite a lot of experience programming online poker, so I said go ahead.

He rang back an hour later and said the company liked my CV, was I available for an interview later in the week. I said yes and gave him a day that I would keep free for the interview. Two days later the recruiter's boss rang and said that he was taking over because the original recruiter had left to become a ski instructor!

The new guy then told me that the boss of the poker company was flying down from Glasgow (to London, where I'm based), and would meet me in a coffee bar in Liverpool Street station. This seemed strange, to put it mildly, but I agreed to the meeting – fortunately I didn't have to carry a copy of yesterday's Times to identify myself, since we now have mobile phones.

I sat in the coffee bar – a sort of narrow corridor squashed into a corner of the station, and my mobile rang. 'We've moved to a better coffee bar upstairs', a voice said, and gave directions. I have to confess that I was starting to get more than a little worried, but I followed the instructions and found the boss and his CTO.

In the event it seemed a fairly genuine affair, the software was demo-ed on everyone's mobile phones and we discussed what needed doing – basically a full audit of the newly developed software to make sure it was fit for purpose. Two days later the recruiter rang and told me they didn't think my poker was good enough for what they wanted...

When the next morning another recruiter rang me and asked if I was interested in a job programming gaming software in Moscow, I decided to gracefully decline the offer.

Might this job suit me?

The next opportunity seemed a little more promising. A Very Big Software Company (aka VBSC) in the city might be interested. They would be sending me a test to take shortly.

The test, when it arrived, turned out to be a URL for one of those BrainBench thingies. I duly answered the questions and it announced that I was an expert in C++, and in the top 10% of people who'd taken the test. This would have been more convincing if I wasn't pretty certain, on the basis of the score it reported, that it got at least one, possibly two of the answers wrong.

Anyway, this gained me an interview (the first of six as it happened). This first one was a phone interview with HR and having passed that I was invited to the VBSC building in the City. At this stage my wife intervened and decreed that I had to wear a suit. I was hauled off to a big department store in Oxford Street and told what I was going to wear. Those of you who know me will appreciate just how traumatic the whole business was.

The first interview was going quite well, although they didn't seem to be asking any high level C++ programming questions.

Then it happened.

One of the guys leaned forward with an intent look on his face and said, 'How would you reverse the words in this.' 'This' was a piece of paper with a sentence of a dozen or so words in it.

'Oh', I said breezily, 'just load the words into an STL container and then take them out in reverse order.' I was about to continue that a stack would probably be the most efficient when I realised that the interviewers had all frozen with stunned looks on their faces. There was a silence. It seemed to stretch on and on, but it probably only lasted a few seconds.

The guy who asked the questions recovered first, and shook himself. 'Ah', he said, 'but supposing you don't have access to the Standard Library?' I have to confess that I nearly blew it then and there. I opened my mouth to laugh and ask what compiler they were using that didn't ship with the standard library, but fortunately managed to restrain myself.

I managed, after a bit of a struggle, to come up with some massively sub-optimal solution which didn't take advantage of any of the useful facilities provided by C++. Doing it the hard way seemed to appeal to them.

Sadly, this set the tone for all the other interviews as well, though it did take a further five sets of interviewers asking questions before they decided that my C++ wasn't bad enough for them!

So, I hung up my suit with a feeling of relief and it was Xmas and the recruiters softly and suddenly vanished away... until the new year.

To be continued... ■

(Note: names have been changed to protect the guilty – not to mention me)

ALAN LENTON

Alan is a programmer, a sociologist, a games designer, a wargamer, writer of a weekly tech news and analysis column, and an ocassional writer of short stories. None of these skills seem to be appreciated by putative employers...

```
while (you care about code)
{
    read ( cvu && overload );
}

do(it);
```

ExCEPTIONAL REWARDS, CENTRAL OxFORD

Why does a leading Quant Fund want to recruit leading software engineers?

You're developing software that's highly efficient and never fails. We're using software to generate positive returns in chaotic financial markets. We think we should talk.

Who Are We?

OxFORD ASSET MANAGEMENT is an investment management company situated in the centre of Oxford. Founded in 1996, we're proud of having generated positive returns for our investors each year, including 2008, especially as many assets are managed for pensions, charities and endowments. We blend the intellectual rigour of a leading research group with advanced technical implementation. We like to maintain a low profile and avoid publicity. Nobody comes to work in a suit and we are a sociable company, enjoying an annual ski-trip away together.

What Do We Do?

We use quantitative computer-based models to predict price changes in liquid financial instruments. Our models are based on analyzing as much data as we can gather and we actively trade in markets around the world. As these markets become more efficient, partly because of organizations like ours, we must have new insights and develop improved models in order to remain competitive. Working to understand and profit from these markets provides many interesting mathematical, computational and technical challenges, especially as markets become increasingly electronic and automated. We enjoy tackling difficult problems, and strive to find better solutions.

Who Do We Want?

Although most of us have advanced degrees in mathematics, computer science, physics or econometrics from the world's leading universities and departments, we are just as interested in raw talent and will consider all outstanding graduate applicants. We expect all prospective candidates to work efficiently both in a team environment and individually. We value mental flexibility, innovative thinking and the ability to work in a collaborative atmosphere. No prior experience in the financial industry is necessary. We want to hear from you if you are ambitious and would relish the challenge, opportunity and excellent compensation offered.

Software Engineers

We are seeking outstanding software engineers to develop and maintain system critical software running in a 24-hour production environment. You will be responsible for all aspects of software development on a diverse range of projects, such as automating trading strategies, integrating third party data into our system and the development of data analysis tools. You will have the following:

- A high quality degree in computer science or related discipline
- C++ experience that demonstrates your ability to work efficiently in a fast paced environment
- Extensive knowledge of the development life cycle, from prototyping and coding through to testing, documentation, live deployment and maintenance

Desirable experience includes Linux, scripting, working with large numerical data sets, and large scale systems.

Systems Administrator

We are seeking a creative, hands-on system administrator who loves dealing with computer hardware to build upon our existing facilities and help us grow into a brand new facility. As our operations grow, we need to manage and process increasingly large volumes (terabytes) of data.

You will be responsible for the day to day monitoring and maintenance of the data centre, as well as determining short and long term resource requirements. You will also design and build a hot standby site, setup and monitor third party connectivity, and scale the entire operation to remote presences in the major world financial centres. You will have experience in the following:

- Project planning
- Linux system administration using Kerberos, LDAP, and Apache
- Setup and maintenance of NAS appliances and backup systems
- High availability systems
- Grid and clustered computing

How to apply

Please email or post CV with covering letter to:

Address
Dr Steven Kurlander,
Oxford Asset Management,
13-14 Broad Street,
Oxford OX1 3BP
United Kingdom

Email
accu2009@applytooxam.com

Telephone +44 1865 258 138

Closing Date
Ongoing

Start Date
Spring 09 – ongoing

Benefits
Health Insurance (including family),
Pension Scheme,
Life Insurance.

OxFORD
ASSET MANAGEMENT

Developer Categorization of Data Structure Fields (Part 2)

Derek Jones, David Binkley and Dawn Lawrie complete their investigation into how developers create data types.

This article is the second of two that investigates patterns in the organization of structure data types. The first part looked at patterns in a large number of **`struct`** definitions extracted from some large programs (e.g., Mozilla and Linux) written in C. This second part analyses the results of an experiment involving professional software engineers performed at the 2005 and 2008 ACCU conferences. This analysis of source code and this experiment are a first step towards understanding the decision making process developers go through when creating the data types used to represent information present within application software.

Experience shows that unless the number of solutions to a given problem is heavily constrained different people will often generate completely different solutions. For even the smallest collection of information there are many different data structures that could be used to represent it. One of the aims of the experiment described in this article is to investigate the extent to which different developers make similar decisions when creating data structures to represent the same information. Another aim is to investigate the impact that the specification has on the ordering of attributes (i.e., fields) in definitions.

The experiment performed at the two conferences asked subjects to create the data structures part of an API that could be used by programs to manipulate various items of information. The following is a list of some of the decisions that subjects need to make when creating this API:

- the items of information (i.e., the fields) contained in the same structure definition,
- the ordering of fields within a single data structure (e.g., a **`struct`** type).
- the name of a field and its type,
- the supporting scalar types that need to be defined. Examples include enumerated types or names given to scalar types (e.g., through use of **`typedef`**). While this is an interesting topic there was not sufficient experimental data to perform any meaningful analysis and it is not discussed further.

The issues discussed in the article are not usually covered by API coding guidelines, which tend to deal with issues such as naming conventions, error handling techniques, default parameters and which default constructs should be defined.[3]

For even a small API there is a huge number of combinations of possible structure type definitions. For instance, there are 2^{10} ways in which the 10 items of information could be split between two structure definitions, 3^{10} between three definitions and so on. The hypothesis that underlies this experiment is that subjects' decisions will be effected by their past experience (e.g., lifetime experience of interacting with the items appearing in the problems and programming experience creating and using data types) and the way in which information is presented in the API specification. The result of shared semantic influences is that patterns will exist in the definitions created by different subjects. This article attempts to uncover and offer explanations for some of these patterns.

Psychology studies

The general problem of how people go about the process of clustering items into categories (which is one way to think about how fields are organized into structure definitions) has been extensively studied by psychologists. A brief overview of some of the key findings from these studies is given in this section. To begin with, children as young as four have been found to use categorization to direct the inferences they make,[4] and many different studies have shown that people have an innate desire to create and use categories (people have also been found to be sensitive to the costs and benefits of using categories[11]). By dividing items in the world into categories of things, people reduce the amount of information they need to learn[15] by effectively building data structures that enable them to lookup information on an item they may not have encountered before, assigning that item to one or more categories, and extracting information common to previously encountered items in those categories. For example, which attributes of an item are used for lookup and how attributes are compared are some of the issues studied by psychologists. For instance, a flying object with feathers and a beak might be assigned to the category *bird*, which suggests additional information such as it lays eggs and may be migratory.

Of necessity the description of items of information in a specification will name objects. To what extent do the names people use for objects effect the categories they are assigned to? For instance, in English the name given to a large stuffed seat for one person is the same as that for a wooden chair, while in Chinese such a large stuffed seat is given the same name as a large stuffed seat for two or more people. In a cross language study where the subjects were native speakers of either English, Chinese, or Spanish[12, 13] Malt, Sloman and Gennari showed subjects pictures of objects of various shapes and sizes that might be capable of belonging to either of the categories— bottle, jar, or container, the subjects were asked to name the objects and also to group them by physical qualities. The results found that while speakers of different languages showed substantially different patterns in naming the objects, they showed only small differences in their perception of the objects (i.e., the categories based on physical attributes).

Category formation

How categories should be defined and structured has been an ongoing debate within all sciences. For instance, the methods used to classify living organisms into family, genus, species, and subspecies has changed over

DEREK JONES
Derek used to write compilers that translated what people wrote. These days he analyses code to try and work out what they intended to write. Derek can be contacted at derek@knosof.co.uk

DAVID BINKLEY

Dave Binkley is a Professor of Computer Science at Loyola College in Maryland. His current research interests include semantics-based software engineering tools, the application of information retrieval techniques in software engineering, and improved techniques for program slicing. David can be contacted at binkley@cs.loyola.edu

DAWN LAWRIE

Dawn Lawrie is an assistent professor at Loyola College in Maryland. Her research interests include the organisation of information and applying information retrieval techniques to software engineering. Dawn can be contacted at lawrie@cs.loyola.edu

the years (e.g., most recently acquiring a genetic basis). Culture (i.e., the society in which a person grew up) has also been found to influence this process. [14]

What method do people use to decide which, if any, category a particular item is a member of? The following are some of the different theories that have been proposed (see the books listed in 'Further reading' for more details):

- The defining-attribute theory proposes that members of a category are characterized by a set of defining attributes. This theory predicts that attributes should divide objects up into different concepts whose boundaries are well defined. All members of the concept are equally representative.
- The prototype theory proposes that categories have a central description, the prototype, that represents the set of attributes of the category. This set of attributes need not be necessary, or sufficient, to determine category membership. The members of a category can be arranged in a typicality gradient, representing the degree to which they represent a typical member of that category.
- The exemplar-based theory of classification proposes that specific instances, or exemplars, act as the prototypes against which other members are compared. Objects are grouped, relative to one another, based on some similarity metric. The exemplar-based theory differs from the prototype theory in that specific instances are the norm against which membership is decided. When asked to name particular members of a category, the attributes of the exemplars are used as cues to retrieve other objects having similar attributes.
- The explanation-based theory of classification proposes that there is an explanation for why categories have the members they do. For instance, the biblical classification of food into clean and unclean is roughly explained by saying that there should be a correlation between type of habitat, biological structure, and form of locomotion; creatures of the sea should have fins, scales, and swim (sharks and eels don't) and creatures of the land should have four legs (ostriches don't). From a predictive point of view, explanation-based categories suffer from the problem that they may heavily depend on the knowledge and beliefs of the person who formed the category; for instance, the set of objects a person would remove from their home while it was on fire.

Threats to validity

For the results of this experiment to have some applicability to actual developer performance it is important that subjects work through problems at a rate similar to that which they would process source code in a work environment. Subjects were told that they are not in a race and that they should work at the rate at which they would normally process code. However, developers are often competitive and experience from previous experiments has shown that some subjects ignore the work rate instruction and attempt to answer all of the problems in the time available. To deter such behavior during this experiment the problem pack contained significantly more problems than subjects were likely to be able to answer in the available time (only one person got as far as attempting the last problem).

Developers often have the opportunity to interact with (e.g., ask questions of) people including the following: domain experts who wrote the specification, likely users of the API, and the paying customer. There was no such opportunity in this experiment; thus, subjects only had their own interpretation of the specification with which to work.

Developers often have the opportunity to study how information is going to be accessed by applications. This allows the data structures to be optimized for common access patterns or to make writing the code simpler (e.g., by reducing the number of arguments that might need to be passed). These design decisions are often made via an iterative process that may involve interaction with the code that uses them.

The creation of data structures is usually an iterative process and the answers provided by subjects in this experiment can only be regarded as a first iteration. Each iteration requires a lot of time and it is not known if subjects considered it more important to answer multiple problems than provide a more refined answer to a smaller number of problems.

Subject experience

Traditionally, developer experience is measured in number of years of employment. In practise the number of lines of source code read and written (interaction with source code overwhelmingly occurs in its written, rather than spoken, form) is a more accurate measure of source code experience. Developer interaction with source code is rarely a social activity (a social situation does occur during code reviews), and the time spent on these social activities is probably small enough to ignored. The problem with the read/write measure is that it is very difficult to obtain reliable estimates of the amount of source read and written by developers. This issue was addressed in studies performed at previous ACCU conferences.[7–9]

What is the relationship between lines of source code written by a developer and the number of data structures they have created? Modifying existing code is a common developer activity. This is likely to result in existing data structures being modified rather new ones created. Your authors do not know of any empirical data that shed light on the relationship between lines of code written and data structure created.

A persons background knowledge has been found to effect the categories they create.[6] Experience shows that it is not unusual for the creator of an API to be a non-expert in the domain being addressed. For consistency it would be advantageous if all subjects taking part in the experiment were either domain experts or not domain experts. No information on subject experience with the domains used in the problems is available.

Subject motivation

Subjects are presented with a list of various kinds of information and have to rely on their experience to design data structures to be used by programs that process this information. Subjects have to estimate the trade-offs among a large number of possible different structure definitions that could be created. At the two extremes are a single structure containing all of the information could be created and separate structure definitions for each piece of information. The following are some of the factors that might play a significant part in the trade off process:

- Simplicity. Writing code is simplified when all required information is obtained by accessing a single structure.
- Information hiding. It may be necessary to restrict access to some of the information, perhaps for commercial or security reasons, or to reduce the likelihood that a fault in one part of a program can effect unrelated data in another part. Distributing the information across different structures is a means of organizing code that helps to minimise the information available at given points in a program.
- Minimising the impact of future updates, with the aim of making it easier to change one part of a system without having to update unrelated parts. For instance, if all of the information was held in a single structure then when a new field is added or an existing field changed, all of the code that referenced the structure would need to be recompiled. If the information was organized into separate structures based on semantic relatedness then adding a new field or changing an existing field, means that only code referencing that structure needs to be investigated when considering what modifications need to be made.
- Minimising the amount of storage used during program execution. For instance, consider a tree whose nodes are represented by a large single structure where each node contains some fields representing frequently accessed information and other fields representing seldom accessed information. In this case, dividing the single structure into multiple structures can yield considerable storage and performance savings,[1] with frequently accessed fields allocated to

one structure and the less frequently accessed information in different structures.

Experimental setup

The experiment was run by one of your authors during a 40 minute lunch time session at each of the 2005 and 2008 ACCU conference (http://www.accu.org) held in Oxford, UK; between 250 and 300 professional developers attend this conference every year. Subjects were given a brief introduction to the experiment, during which they filled in background information about themselves, and they then spent 30 minutes (2005) and 20 minutes (2008) working on the problems. All subjects volunteered their time and were anonymous.

The problem

Figure 1 is an excerpt of the text instructions given to subjects.

The experiment brochure handed out to subjects briefly specified three different kinds of domain specific information, each of which followed the format given in the Example:

- Department of Agriculture. This involved: Meat (cattle, sheep, and pigs) and Cereals (barley, corn and wheat) and the following list of items: Owner of farm, Farm acres used to grow crops, Seed supplier, Location of field, Fertilizer costs, Weed killer sprayed, Fertilizer applied, Date crop harvested, Weight of average shipment, Was organically produced, Number of each kind of animal on farm, Acre of crop market value, Farm location, Antibiotics used, Feed-stuff costs, Market value of kind of animal, Genetic ID, Crop sown date, Last vet inspection date, Average labor cost for growing crop, Farm acres used to rear animals, Pedigree ID, Date animal slaughtered, Date animal born and Animal raising average labor cost. Some subjects saw a subset containing 15 items, where some items had been combined into a more generic description (e.g., 'Crop sown date' and 'Date animal born' becoming 'Date agricultural product started life').
- Department of Transport. This involved: Self driven (Walking, Cycling, Car and Motor bike) and Passenger (Ferry, Train, Bus, and Flying) modes of transport and the following list of items: Arrival time affected by congestion, Weekly travel cost, Method of transport is environment friendly, Travel distance, Yearly expenditure on infrastructure supporting transport method, Workforce percentage using transport method, Travel time, Tax revenue from transport method and Probability of arriving on time.
- Department of Natural Resources. This involved: Industrial stone (Marble, Slate, and Granite) and Purified metal (Gold, Silver, and Titanium) and the following list of items: Raw material per shipment, average weight, Production volume per week, Mining location, Raw material value on commodity exchange, Storage cost per week, Cost of extraction per unit weight, Mining date of raw material and Dispatch method (e.g. sea/air).

Variations on the problem

There are various ways in which the problem information seen by subjects could be organized. The format of the experiment meant that the organization had to be kept as simple as possible. The following describes the information organization that was varied:

- The order in which items of information were listed in the problem specification. For instance, in the Example problem the specification uses the order 'Name of given town' followed by 'Number of bathrooms'. The order in which items were listed was randomly chosen for each printed set of problems.
- In many cases it is possible to use different phrases to describe an item of information. For instance, the phrase 'Parking space included with accommodation' can also be worded as 'Accommodation has parking space included with it'. In most cases the phrase seen by each subject was randomly selected from two possibilities (in a few cases only one phrase was reasonable).

Results

Approximately 240 (2005) and 300 (2008) people attended the conference, of which 11 (2005) and 15 (2008) took part in the experiment. The 26 subjects produced 38 answers (Agriculture: 22 Transport: 12 Natural Resources: 1. (Three subjects gave an answer to the example problem.) Some answers were incomplete at the end of the allotted time. On average each participant completed 1.5 problems. The average number of years of experience of subjects was 11.1 (standard deviation 6.3). Subjects were told that they could use any computer language and the languages used were (subject counts in parenthesis) C++ (15), C (8), Java (1), C# (1) and Python (1).

For the Agriculture problem, the average number of structure definitions created per answer was 5.2 (standard deviation 2.5), and these definitions contained on average of 5 fields (sd 3.5) (this includes fields not found in the *specification*).

In the following discussion of the study's results the term *specification* is used to refer to the description of an item as found in the question seen by a subject (e.g., 'Size of garden'), while the term *field* will be used to denote the name given by the subject (e.g., '`garden_size`'). The term *component* is used to refer to a word or abbreviation found within a field. *Components* might be separated using underscores or camel casing. For example, '`garden_size`' includes two components: '`garden`' and '`size`'.

Subjects sometimes wrote comments on their answer sheets that were obviously intended to be directed at those running the experiment rather than source code comments that were part of an answer to the problem. The common comment was that there was insufficient time to complete an answer.

Influence of specification on information ordering in declarations

The information in a specification is ordered in various ways and it is hypothesized that this ordering has an effect on how developers order various entities within a definition. The following are the information ordering issues analysed in this subsection:

- the effect that the order in which items of information are listed in a specification has on the order in which fields are defined in a structure. For instance, if the specifications lists the 'number of bathrooms' before the 'number of bedrooms' the field **`number_of_bathrooms`** may be expected to appear before **`number_of_bedrooms`** (assuming they both appear in the same definition and that these names are used).
- the effect that the words used and their ordering in each specification has on the components of a field. For instance, the specification 'number of bedrooms' might be expected to result in the field **`number_of_bedrooms`** while the specification 'bedroom count' might be expected to result in the field **`bedroom_count`**.

Subjects sometimes gave answers that did not contain some items of information. This is likely to have been an oversight, although at least one subject made an explicit decision (their answer contained a comment explaining how some items could be deduced from information on other items, e.g., organically produced from information on the fertilizer/weedkiller used).

Some subjects gave answers that included fields that did not directly correspond to an item listed in the specification (e.g., minimum/maximum/mean/standard deviation arrival delays in the Transportation API). Such fields were ignored in the analysis.

Item words in field names

Identifiers are often formed by combining two or more known words. If subjects are influenced by the content of a specification it is to be expected that the words used and the order in which they occur will be reflected in the name of the corresponding field. For instance, given the description 'Fertiliser costs' a field name of **`fertiliser_costs`** or `fertiliser_cost` might be expected, while the specification 'Cost of fertiliser' might result in the field name `cost_of_fertiliser` (perhaps with the `of` omitted) or `fertiliser_cost`.

It is possible that subjects have relatively fixed naming habits or that the given specifications induced subjects to use a particular naming. The results were therefore analysed by subject and by item of information as follows:

1. The name of a field is broken down into its components (e.g., `bathrooms_count` has the components `bathrooms` and `count` as does `bathroomsCount`). The algorithm is the same as that used in a previous ACCU experiment.[9]

Figure 1

You will be asked to define some data structures to hold values for various kinds of information. This is not a race and there are no prizes for providing answers to all questions. If you do complete all the questions feel free to add any additional comments to your previous answers.

Yumei is a medium sized island that gained independence in 1945. The island economy is based on the export of a various kinds of raw materials and agricultural products. The government has decided to publish a set of APIs that applications written for government departments must follow.

The government could not reach consensus on an object oriented approach and it was decided to specify data types and access functions separately. You have been assigned the job of creating the data structures that will be used to hold various kinds of information. If possible please briefly specify any rationale used to decide how to organise the data structures.

The declarations can be written in a language of your choice (it would simplify subsequent analysis if either C or Java were used). The members may have any arithmetic type, or a pointer type, or an array type.

1. Read the information that needs to be represented.
2. Decide which information should be held in a given data structure.
3. Decide on a type to be used to represent the value of the information.
4. Decide on the name of a member to be used to denote the information.
5. Define one or more structure types to hold members denoting the information.

The following is an example question

The Department of Housing maintains rented accommodation for workers. The form of accommodation can be one of:

Bungalow
Caravan
Dormitory
Flat
House
Youth hostel

The information, about the accommodation, that needs to be processed by applications includes the following (in alphabetical order):

1. Address of accommodation
2. Detached, Semi-detached, Terraced
3. Easily moved
4. Floor number of accommodation
5. Name of current responsible occupant
6. Name of given town
7. Number of bathrooms
8. Number of bedrooms
9. Number of each kind of accommodation in given town
10. Number of rooms
11. Parking space included with accommodation
12. Shared bathroom
13. Size of garden
14. State of repair
15. Total area of floor space
16. Total value of each kind accommodation in given town
17. Weekly rent

```
#define HOUSE 0
#define BUNGALOW 1
#define CARAVAN 2
#define FLAT 3
#define DORMITORY 4
#define YOUTH_HOSTEL 5
#define MAX_ACCOM_KIND 6

#define NOT_APPLICABLE 0
#define DETACHED 1
#define SEMI_DETACHED 2
#define TERRACED 3

struct town_accommodation {
  char * town_name;
  int num_accom_kind[MAX_ACCOM_KIND];
  float value_accom_kind[MAX_ACCOM_KIND];
  };

struct accommodation {
  int kind_of_accommodation;
  int num_rooms;
  int num_bedrooms;
  int num_bathrooms;
  float garden_size;
  float weekly_rent;
  int detached_status;
  char easily_moved;
  char shared_bathroom;
  char has_parking_space;
  int floor_number;
  char * address;
  char * repair_state;
  char * current_occupants;
  };

typedef unsigned char BOOL;

struct caravan_rec {
  int number_of_rooms;
  int number_of_bedrooms;
  int number_of_bathrooms;
  float size_of_garden;
  float rent_per_week;
  BOOL bathroom_is_shared;
  BOOL parking_space_available;
  int floor_number;
  char * address;
  char * state_of_repair;
  char * current_occupiers;
  };

struct accommodation_rec {
  int accommodation_kind;
  int number_of_rooms;
  int number_of_bedrooms;
  int number_of_bathrooms;
  float size_of_garden;
  float rent_per_week;
  int detached_status;
  BOOL bathroom_is_shared;
  BOOL parking_space_available;
  int floor_number;
  char * address;
  char * state_of_repair;
  char * current_occupiers;
  };
```

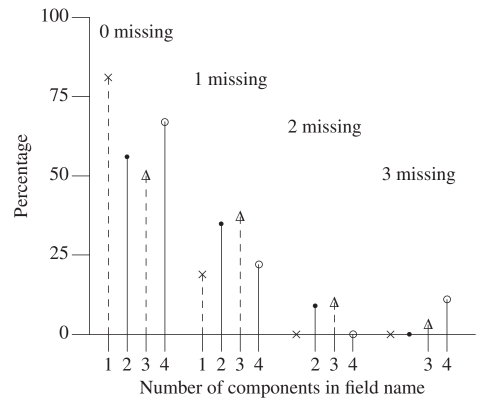
2. For each component of a field what was not a dictionary word, the component was treated as an abbreviation and replaced to the appropriate dictionary word. Based on the specification, there was always a unique replacement of each abbreviation. For example, **`num`** was replaced with **`number`**.
3. The components from each field were compared to the words from the corresponding specification. A component and a word match if they have the save *stem* (e.g., the stem of 'swimming' is 'swim'). Stemming was performed using the Krovetz stemming algorithm.[10] This comparison was carried out twice: once ignoring the order of the component and word and the second time requiring the same relative order.

Counting the number of components that appear in the corresponding specification, in almost 70% of the cases all the components of a field occur in the specification of the corresponding item. Figure 2 gives a break down by fields containing 1, 2, 3 or 4 components and the percentage where the corresponding specification does not contain zero or more of these components.

The 50 cases, out of a total of 198 unique components, where a component did not appear in the corresponding specification can be broadly broken down into the following categories:

- the component is more specific: there may be several fields each relating to some aspect of the specification, e.g., an address specification includes information on street, postal-code, etc.). Another example is the fields **`garden_length`** and **`garden_width`** created to hold information on 'Size of garden'.

Figure 2



Percentage of fields containing a given number of components (e.g., 1, 2, 3 or 4) where the field's components also occur in the corresponding specification. In the group of four on the left all components occur in the specification; in the next group one component does not occur in the specification; and so on. Combined data from 2005 and 2008, a total of 260 fields having 704 components.

- synonym-like usage: for example *type* used in place of *kind* and *count* used in place of *number*, leading to the choice of the field name **`product_type_count`** for 'Number of each kind of agricultural product'.
- grammar support words: words such as *is* in **`is_organic`** and *of* in **`cost_of_feed`** are used to generate short phrases.

Having shown that a large percentage of components occur in the specification we now compare the relative ordering of components and the words of the specification. This ordering might be effected by the following.

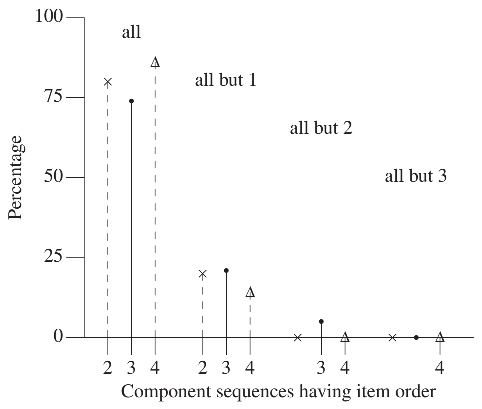
- An existing set of rules that the subject always uses to create fields from combinations of words.
- The relative word order present in the corresponding specification.
- A desire for consistency in all field names. The specifications seen by some subjects have characteristics that require decisions to be made about conflicting choices.

The description of many items was randomly chosen from a small set of possibilities; thus in some cases the descriptions given for two semantically related specifications use different word order patterns. For instance, a subject may see the two specifications 'Weedkiller costs' and 'Cost of fertiliser' (rather than 'Fertiliser costs' in the later case).

Subjects who base their choice of field name purely on the wording that appears in the corresponding specification will not pay any attention to the word order found in the specification. However, subjects trying to achieve a degree of consistency across semantically related fields may select one kind of word ordering over another, perhaps the word order from the description that is first encountered (likely the first one listed in the list of items).

Figure 3 breaks down the fields that contain 2, 3 or 4 components based on the number of components occurring in the same order as the corresponding words from the corresponding specification (a total of 260 fields). The figure omits fields having only one component in common and also omits the single data point of one field with 6 components (because patterns cannot be established from a single data point). There were no fields with 5 components..

Figure 3



Percentage of fields containing a given number of components (e.g., 2, 3 or 4) where the field's components occur in the same order as in the corresponding specification (other words may appear in the specification between matching components). In the group of three on the left fields containing 2, 3 and 4 components all occurring in the same order as the corresponding specification, in the next group of one component does not occur in the same order and so on. Combined data from 2005 and 2008, a total of 260 fields.

Of these 260 fields, the components occur in the same order as the words of the corresponding specification in 86% of cases(37 cases did not follow this order). This provides strong evidence for the hypothesis that the subjects use the ordering of information in the specification when creating field names.

Table 1 lists some examples of fields definitions given as answers to a given specification. The samples are grouped under four common patterns that are apparent in the data..

Is there a pattern to the 14% of cases where the specification order was not following? One possible pattern is a preference for what might be considered to be *natural word order*. There are many patterns to English word order, with some orders being much more common than others. Perhaps subjects made use of their knowledge of what they consider to be *natural* word order. To test for this pattern of usage Google's n-gram dataset[2] was used; this data set includes counts of word pairs, triples, etc. as found by Google on the web.

The frequency of occurrence of components in each field and the corresponding specification word order were looked up in the Google dataset. For example, the components **`start date`** occur 872,156 times, while the words of the corresponding specification 'date started' occur only 4,690 times. Thus the order of the components used by the subject in this answer is 186 times more common, in the Google observations, than the ordering used in the specification.

A breakdown of the 37 data points is presented in Table 2. In the table 'No observation' include the 32.5% cases in which neither the components of the participants field nor the words of the specification occurred in the

Table 1

Some wording patterns and examples of them that appeared in subject answers. The “consistency with previous field” pattern occurs when the name of a field uses the same component ordering as that used by the field that occurred before it in the structure definition.

	Field	Specification
type name first	SCurrency currencyRevenue STime timeTravel	Transport method tax revenue Travel time
favour preposition	float costPerWeek float probabilityOnTime float revenueOfTax float percentageOfUsers	Cost of travel per week Probability of arriving on time Tax revenue from transport method Workforce percentage using transport method
consistency with previous field	int costs_material int costs_labor	Costs (e.g., Feed-stuff and Fertilizer) for agricultural product growing Agricultural product average labor costs
noun adjective	boolean bathroomShared boolean parkingSpaceIncluded String occupantName	Shared bathroom Parking space included with accommodation Name of current responsible occupant

Table 2

Categorizes the fields components that appear in to different ordering based on observations in the Google n-gram dataset.[2] No observations - Number of field component sequences that did not appear in the Google n-gram database. Subject more common - The sequence used by the subject was more common than the specification. About the same - both sequences had similar number of occurrences, and Specification more common - the specification was the more common case.

	Count	Percentage
No observations	12	32.5
Subject more common	13	35.0
About the same	5	13.5
Specification more common	7	19.0

Google dataset. The row ‘Subject more common; represents cases in which the component order was at least 60% more common in the Google dataset that the specification word order. Similarly, ‘Specification more common; represents cases in which the specification word order was at least 60% more common in the Google dataset that the component order. The final category ‘About the same; includes the remaining cases.

The third of the data for which no observations were found all have three or more components that represent phrases rarely seen in the Google dataset, for example, ‘life start date’.

At 35% use of the more common order was almost twice as likely as using an order that was less common than that appearing in the specification. Possible reasons for this usage is a desire by subjects for consistency of component order for related fields in the same structure definition.

Ordering of fields

Does the order of fields in structure definitions tend to follow the order in which the corresponding item of information appears in the specification? For instance, in the Example problem given earlier four items had the relative order: *Number of bathrooms*, *Number of bedrooms*, *Number of rooms* and *Shared bathroom*, and their corresponding fields had the relative order **`num_rooms`**, **`num_bedrooms`**, **`num_bathrooms`** and **`shared_bathroom`** (the two lists almost have opposite orders).

There are 24 possible ways this information can be represented using four fields (see Table 3). In a random selection every field ordering has a 1 in 24 probability of occurring, but what is the probability of a random field ordering matching that of the corresponding words listed in the specification?

To better understand this question, we count the *field-item agreement* for a sequence of fields. This metric represents the number of pairs that are in the expected order. For instance, the sequence 2314 is assigned the value 4: 2 appears before 3 and 4, 3 appears before 4 and 1 appears before 4. There are five sequences having an agreement metric of 4, see Table 3, and the probability that a random sequence of four fields will have an agreement metric of four is 5=24.

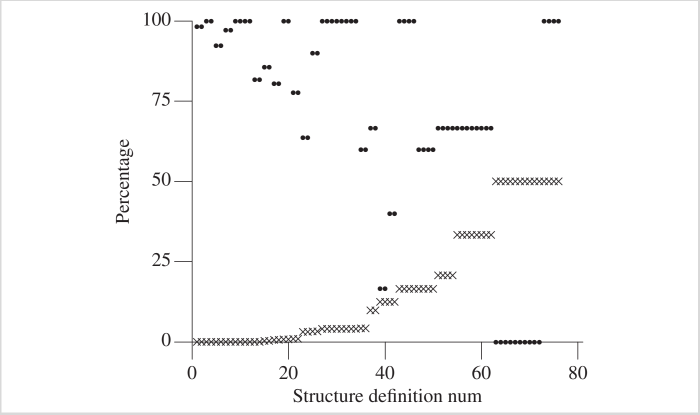
It was noted that the sequence 1; 3; 5; 6; 5; 3; 1 (second column in Table 3) is part of the Triangle of Mohanian numbers – sequence A008302 in the Encyclopedia of Integer Sequences, www.research.att.com/~njas/sequences – and also appears in the number of inversions of a permutation.

Table 3

The 24 possible ways in which four fields can be ordered. Rows have the same field-item agreement metric (the extent to which a field ordering matches that of the corresponding items in the specification, the higher the better).

Metric	Occurrences	Orders							
6	1	1234							
5	3	1324	2134	1243					
4	5	2314	3124	1342	2143	1423			
3	6	3214		2341	3142	1432	2413	4123	
2	5			3241		2431	3412	4132	4213
1	3					4321		4231	4312
0	1							4321	

Figure 4



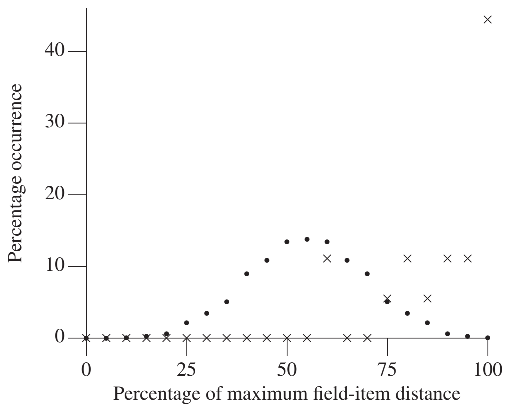
Probability that the measured field-item distance, for a given structure definition, could occur through random selection of fields (crosses) and the same structure’s field-item distance expressed as a percentage of the maximum possible (bullets). If field order selection was random the bullets would be distributed around 50%. The structures measured are ordered, left to right, by increasing probability of field-item distance.

This information was used to derive the possible number of structure definitions having a given field-item agreement metric for definitions containing larger numbers of fields.

Figure 4 shows the order distributions for the random case and the collected data. In the figure crosses at 50% indicate that there is a 50/50 change of the fields having the order they do (i.e., there are two fields in the definition). Moving left the crosses at 33.3% denote combinations involving three fields (there are also crosses at 16.67% for less likely combinations of three fields), crosses at 20.8% for some combinations of four fields, and so on. The crosses very close to zero are for definitions containing the less likely combinations of 6 or more fields, or just containing lots of fields.

If field order selection was random then on average the field-item distance would be 50% of the maximum value. The bullets in Figure 5 show the expected distribution of field-order distance that would occur with random field ordering. The crosses denote the percentage occurrence found in problem answers (answers with greater than 5% chance of occurring randomly were ignored). A significant percentage (44.4%) had the maximum field-item distance (i.e., the relative field order was the same as the items in the specification).

Figure 5



Bullets represent the probability (y-axis) of fields in a structure definition having a given percentage of the maximum field-item distance (x_axis). Crosses denote structure definitions in problems answers whose field-order is less that 5% likely to have randomly occurred. Counts have been put into bins of width 5 (percent).

Information clustering

There are a lot of different ways in which information can be clustered into different structures. For instance, there are 2^{17} ways in which the 17 values listed in the earlier example could be split between two structure definitions, 3^{17} between three definitions and so on.

The problems subjects were asked to solve have no correct answer as such. If the development history for a variety of different APIs was available such things as development cost, ease of maintenance and the other factors discussed in the subsection on Subject motivation could be compared and those with lower costs and/or greater benefits labelled as being 'better' than the others.

The analysis in this subsection looks for general patterns in how items tended to appear together within the same structure definition.

Finding clusters

The organization of the data (i.e., a list of experimenter provide items clustered into separate structure definitions in a manner decided by each subject) and the analysis required (i.e., averaged over all subjects what clustering of items exists, and how strong is it) is very similar to a study performed Ross and Murphy[16] and the mathematical techniques used in that study are used here. In one of the experiments performed by Ross and Murphy subjects were given a list of various foods and asked to divide them into groups; either 'similar food types', 'foods that are eaten in the same situation' or 'things that go together'.

Ross and Murphy generated a Robinson matrix from the subject data and this was used to obtain information on the general form of the categories created by their subjects. A Robinson matrix has the property that the value of its matrix elements decreases, or stays the same, when moving away from the major diagonal (more mathematical details in the Ross and Murphy article[16]). A matrix can be put into this form by reordering its rows and columns. This reordering brings together entries that are similar and moves dissimilar ones apart. For this study the seriation package[5] within the R statistical program, http://www.rproject.org, was used to create a Robinson matrix. Figure 6 and Figure 7 are visual representations of this matrix for one problem.

The *Department of Agriculture* problem produced the most answers (22); it also contained the largest number of items (25), making it much more likely that answers contain many structure definitions. The analysis in this subsection is applied to the answers to this problem.

The similarity detection process involves the following two stage process, for the answers to the Department of Agriculture API:

1. creating a similarity matrix from the structure specifications. The rows and columns of this matrix both represent the specifications listed in the problem, for instance if the entry for 'Easily moved' is in row 3, then column 3 also denotes 'Easily moved'. The entries for this similarity matrix are calculated as follows: 1) Zero the matrix, 2) for each subject perform the following: a) add 1 to every entry in the matrix where two specifications occur together in the same structure definition (e.g., if a definition contains fields representing both 'Easily moved' and 'Shared bathroom' add 1 to the entry at the corresponding row/column (the matrix is symmetrical, so two entries will be incremented).
2. Within the seriation package the seriate function is capable of processing data having a variety of forms and implements a variety of algorithms that attempt to optimize the process of generating a Robinson matrix from this data (the computational cost is so high that approximations have to be made). It was found that consistent results were obtained by mapping the data to a dissimilarity form. This was done using the formula 1– *max_element*/*max_element_val* for each element of the matrix, where *max_element_val* is the largest value in any element of the matrix.

The R instructions used to generate these images were:

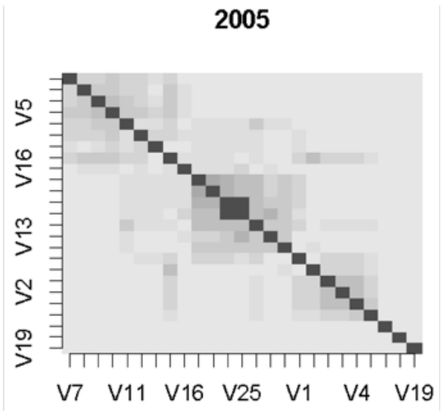
```
library("seriation")
fmat <as.matrix(read.table("f.data"))
fdist <as.dist(1 fmax/max(fmat))
fser <seriate(fdist, method="BBURCG")
pimage(fdist, fser)
```

The items corresponding to the ticks along both axis in Figure 6 and Figure 7 are as follows (the alternative specifications are also listed):

```
1 Date of last vet inspection | Last vet inspection
date
2 Date animal born | Animal birth date
3 Date crop sown | Crop sown date
4 Date animal slaughtered | Animal slaughter date
5 Date crop harvested | Crop harvest date
6 Antibiotics used | Antibiotics given
7 Fertilizer used | Fertilizer applied
8 Weed killer used | Weed killer sprayed
9 Pedigree ID | Pedigree class
10 Genetic ID | Genetic family
11 Supplier of seeds | Seed supplier
12 Was organically produced | Produced organically
13 Number of each kind of animal on farm
14 Farm acres used to grow crops | Acres used to
grow crops
15 Farm acres used to rear animals | Acres used to
rear animals
16 Average shipment weight | Weight of average
shipment
17 Market value of kind of animal
18 Market value of acre of crop | Acre of crop
market value
19 Average labor cost for raising kind of animal |
Animal raising average labor cost
20 Average labor cost for growing crop | Crop
growing average labor costs
21 Feedstuff costs | Cost of feedstuff
22 Fertilizer costs | Cost of fertilizer
23 Location of field | Field location
24 Location of farm | Farm location
25 Farm owner | Owner of farm
```

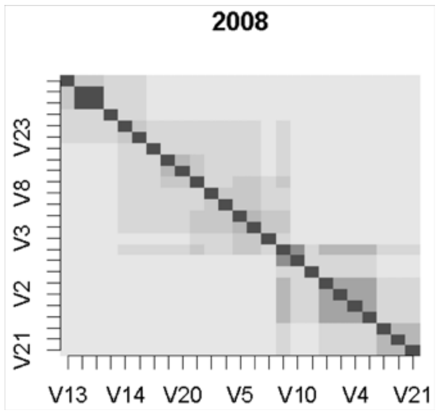
Comparing the clusterings extracted from the 2005 and 2008 answers (Figure 6 and Figure 7) it can be seen that both the 2005 and 2008 data appear to be made up of approximately three large clusters (combining the data from both experiments produces a result that is almost identical to that for 2008). Within each of these large clusters are smaller clusterings of field items.

Figure 6



A visualization of the contents of the Robinson matrix generated from the 2005 field-item structure membership information. Darker areas indicate more closely associated fields. The order of items along the x and y axis is: 7, 3, 8, 5, 11, 22, 23, 12, 16, 15, 14, 24, 25, 21, 13, 17, 1, 10, 9, 2, 4, 6, 18, 20 and 19 (plot shows a subset of these values).

Figure 7



A visualization of the contents of the Robinson matrix generated from the 2008 field-item structure membership information. Darker areas indicate more closely associated fields. The order of items along the x and y axis is: 13, 24, 25, 15, 14, 23, 22, 18, 20, 11, 8, 16, 5, 7, 3, 12, 10, 9, 1, 2, 4, 6, 17, 19 and 21 (plot shows a subset of these values).

The only two items that are consistently placed in the same definition are items 24 and 25 ('Location of farm' and 'Farm owner') and items 14, 15 and 13 are often included in the same definition with them (the name given to definition containing this set of fields was often farm or something very close to this name; no analysis of this kind of type name was made). Other items are often placed in the a definition together the items 2, 4 and 6; other small sets of items can be extracted from the figures.

Comparison with source code measurements

The first part of this article included various measurements of a large amount of C source code. This subsection compares those measurements against the same measurements for the structure definitions given in the problem answers. The number of fields contained in the source code struct definitions varied between 2 and 70 fields, while the definitions given in the answers to the experiment problems contained a small range of fields (between 2 and 15; average 5).

The first part of the article included three figures containing measurements of field attributes and these are discussed in the following:

1. See Figure 8 for a comparison of experiment answers and source code measurements. For those deltas having a zero measured percentage there were no instances of a minimal type change sequence because of the small number of occurrences of a given type sequence (often only one occurrence).
2. In the case of the number of individual field pairs that share one or more subcomponents because of the small number (i.e., 104) of structure definitions available for analysis in the answers there is a great deal of variation the number of occurrences of field-pair distances and it is not possible to make reliable comparisons against the source code measurements.
3. For the problem answers the average distance between field pairs for definitions containing a given number of fields appears to be similar to that found in the source code measurements. The small number of data points means it is not possible to make a stronger statement. For those wanting a comparison against the corresponding figure in part 1 of this article, the actual x/y axis values are: 3.14:5, 1.90:6, 1.00:7, 0.00:8, 1.00:9, 3.67:10, 0.00:11, 0.00:12 and 6.89:13.

Conclusion

This study investigates factors that influence the decisions made by developers when deriving API structure definitions from a specification.

The hypothesis was that various kinds of information in the specification had a significant impact on the way in which structure definitions were organized and the naming of fields. The particular attributes found to have an effect included:

- The ordering of fields was strongly influenced by the order in which items of information occurred in the specification.
- Any words appearing in the components of a field name were likely to also occur in the corresponding specification
- Any words appearing in the components of a field name were likely to either appear in the same order as that in the specification or to follow common English word order.

An analysis of individual structure definitions showed that for some items of information subjects made similar decisions on putting them together within the same structure definition. It is not known whether this clustering would have been stronger if subjects had had time to iterate on their initial design decisions.

Further experiments are needed to confirm the results of this study and to measure the consequences of allowing subjects more time to individual answer questions. ■.

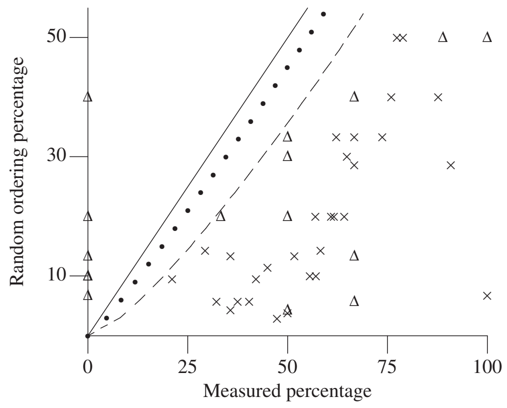


Figure 8

The measured percentage of structure definitions having a minimal type change sequence (x-axis) and the percentage that are expected to occur if fields were ordered randomly. A delta symbol indicates a problem answer percentage and the random percentage for a definition containing that particular number of types for problem answers, while cross applies to source code measurements. If field ordering were random the deltas/crosses would be expected to cluster along the solid diagonal line (bullets indicate 1 standard deviation, dashed line 3 standard deviations). Data for all definitions containing between four and seven (inclusive) fields.

Further reading

A readable upper graduate level book dealing with some of the more higher level aspects of how people create and use categories *The Big Book of Concepts* by Gregory L. Murphy published by MIT Press, ISBN 0-262-63299-3.

This 'Software' Stuff, Part 3

Pete Goodliffe concludes his mini-series on the nature of our craft.

Over the last few articles we've been taking a bird's eye view over the art, the craft, and the science of software development. We're looking at what software is, how it works, and how we build it. The whole point of this exercise is to identify the ways we can each improve as software developers.

We'll wrap up this mini-series by seeing how software is both child's play and a chore. Remember that as we go along I will be posting a series of personal questions. These questions are guides to help you work out specific ways to improve your coding skills. Carefully consider each of the as you read, and work out whether they apply to you. What can you do about them?

Before we dive in, it's worth revisiting the very first question I posed:

Do I... want to improve as a programmer?

In these days of economic hardship and reduced job security, surely you owe it to yourself to improve as a programmer? Make yourself invaluable, and more easily employable! Whether through this article, the other articles in this magazine, or any other mechanism (books, the ACCU 2009 conference, practising) work out how you can grow and develop in this craft.

Software is... child's play

For me, this observation seems particularly appropriate; I'm really just a child at heart. Aren't we all?

My kids are now six and four. It's incredibly interesting to see how they continue to grow and learn, how their world view changes and is shaped by each new experience. We can glean a lot from the way a child learns and reacts to the world. Consider how these apply to our software development:

- **Learning** A child is constantly aware that they are learning, that don't know everything. This requires a simple characteristic: humility. Some of the programmers I have found hardest to work with think that they know it all. If there's something new they need to know, they read a book and then presume that they're an expert. A total humility bypass.

PETE GOODLIFFE

Pete Goodliffe is a programmer who never stays at the same place in the software food chain. He has a passion for curry and doesn't wear shoes. Pete can be contacted at pete@cthree.org

Developer categorization of data structure fields (continued)

A readable collection of papers on how people make use categories to solve problems quickly without a lot of effort: *Simple Heuristics That Make Us Smart* by Gerd Gigerenzer, Peter M. Todd and The ABC Research Group, published by Oxford University Press, ISBN 0-19-154381-7.

Acknowledgements

The authors wishes to thank everybody who volunteered their time to take part in the experiments and ACCU for making a slot available, in which to run the experiment, at both conferences.

Thanks to Matt Hearn, Brittany Babin and Oluwaseyi Fatadi of Loyola College for decrypting subjects' written answers into machine readable form.

Thanks to Michael Hahsler for suggestions on the use of his seriation package.

References

[1] R. W. Bowdidge, 'Refectoring gcc using structure field access traces and concept analysis', in *Procedings of the Third International Workshop on Dynamic Analysis (WODA 2005)*, pages 1–7, May 2005.

[2] T. Brants and A. Franz, *Web 1T 5-gram,* Linguistic Data Consortium, Philadelphia, USA, 1 edition, 2006.

[3] K. Cwalina and B. Abrams, *Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries*, Addison–Wesley, 2006.

[4] S. A. Gelman and E. M. Markman, 'Categories and induction in young children', *Cognition*, 23:183–209, 1986.

[5] M. Hahsler, K. Hornik, and C. Buchta, 'Getting things in order: An introduction to the R package seriation', *Journal of Statistical Software*, 25(3):1–34, Mar. 2008.

[6] E. Heit, 'Background knowledge and models of categorization' in U. Hahn and M. Ramscar, editors, *Similarity and Categorization*, chapter 9, pages 155–178. Oxford University Press, Apr. 2001.

[7] D. M. Jones, 'I_mean_something_to_somebody', *C Vu*, 15(6):17–19, Dec. 2003.

[8] D. M. Jones, 'Experimental data and scripts for short sequence of assignment statements study' http://www.knosof.co.uk/cbook/accu04.html, 2004.

[9] D. M. Jones, 'Experimental data and scripts for developer beliefs about binary operator precedence' http://www.knosof.co.uk/cbook/accu06.html, 2006.

[10] R. Krovetz, 'Viewing morphology as an inference process', Technical Report UM-CS-1993-036, University of Mass-Amherst, Apr. 1993.

[11] W. T. Maddox and C. J. Bohil, 'Costs and benefits in perceptual categorization', *Memory & Cognition*, 28:597–615, 2000.

[12] B. C. Malt, S. A. Sloman, S. Gennari, M. Shi, and Y. Wang, 'Knowing versus naming: Similarity and the linguistic categorization of artifacts', *Journal of Memory and Language*, 40:230–262, 1999.

[13] B. C. Malt, S. A. Sloman, and S. P. Gennari, 'Universality and language specificity in object naming', *Journal of Memory and Language*, 49(1):20–42, 2003.

[14] R. E. Nisbett and A. Norenzayan, 'Culture and cognition' in D. Medin and H. Pashler, editors, *Stevens' Handbook of Experimental Psychology, Volume Two: Memory and Cognitive Processes*, chapter 13. John Wiley & Sons, third edition, Apr. 2002.

[15] E. M. Pothos and N. Chater, 'Rational categories' in *Proceedings of the Twentieth Annual Conference of the Cognitive Science Society*, pages 848–853, 1998.

[16] B. H. Ross and G. L. Murphy, 'Food for thought: Cross-classification and category organization in a complex real-world domain', *Cognitive Psychology*, 38:495–552, 1999.

A child is constantly assimilating new knowledge. We must recognise that if we want to improve, we must learn. And we must be realistic about what we do, and do not, know.

Enjoy learning, savour finding out new things. Practise and improve your craft.

- **Simplicity** Ask yourself: do I write the simplest code possible? Do you reduce everything to the least complex form to make it easier to understand and easier to code? I love the way kids try to get to the bottom of things, to understand things from their own limited perspective. They're always asking why. Take, for example, a recent conversation I had with my six- year-old daughter: Daddy, why is Millie my sister? Because you're in the same family as her, Alice. Why? Well, because you have the same mummy and daddy, Alice. Why? Because, well, you see, there are the birds and the bees... Oh go and get a book! Why?...

  We should be constantly asking why – questioning what we are doing and the reasons for it. Seeking understanding of the problem and the best solution. And we should strive for simplicity in our handiwork. That is not the most simplistic 'dumb' code possible, but appropriately non-complex code.
- **Looking cute** If all else fails, this childish characteristic might help you to get a pay rise. Perhaps more helpful for the 'career' programmer than the 'professional' developer?

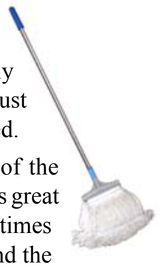
With that in mind, we can ask ourselves:

Do I... write the simplest code possible? Or do I type what comes to mind, and not think about commonality, refactoring, or code design?

...and...

Am I... still learning? What can I learn about? What do I need to learn about?

Software is... a chore



A lot of our software development work is not necessarily pleasant. It's not glamourous. It's not plain sailing. It's just donkeywork that has to be done to get a project completed.

To be an effective programmer, you mustn't be afraid of the chores. Recognise that programming is hard work. Yes, it's great to do the cool design on newest product version, but sometimes you need to do the tedious bug fixing and grubbing around the old awful messy code to get a product shipping and make some money.

From time to time we must become software janitors. The requires us to:

- **Clean up** We must spot problems and address them; work out where breakages are and what the appropriate fixes are. These fixes must be made in a timely and non-disruptive manner. A janitor does not leave the unpleasant tasks to someone else, but takes responsibility for them.
- **Work in the background** A janitor does not work in the limelight. They probably receive little recognition for their heroic efforts. This is very much a supporting, not a lead role.
- **Maintenance** A software janitor will remove dead code, fix broken code, refactor and rebuild inappropriate workmanship, and tidy and clean what is OK to ensure that it doesn't fall into disrepair.

Am I... happy to do code 'chores'? Do I only want the glamourous work?

...and...

Do I... take responsibility for messy code and clean it up?

The ideal programmer revisited

I built a composite image of the ideal programmer based on the observations that software is:

- An art
- A science
- A sport
- Child's play
- A chore

And got this. That's definitely something to aspire to.

But thinking about it, Van Gogh was a bit of a plank; he cut off his own ear, after all. Einstein was as sharp as a button, but clearly a few electrons short of an atom; a few particles short of his wavelength. Just look at the hair. Sportsmen are intentionally bonkers – running around on a cold muddy pitch inflicting pain and torture on themselves in the name of 'enjoyment'. Nutters. And janitors? When I was at school, I was convinced that they're all evil geniuses plotting to take over the world from their underground boiler rooms.

Frankly, they're all a little bit 'abnormal', aren't they?

So my observation is this: programmers are simply not a socially well-adjusted bunch. You have an excuse. So go forth and be... interesting.

A final thought

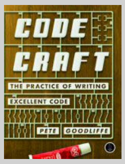
At the beginning of this series (remember all the talk of spaghetti and custard?) we complained that too much software is like Alphabetti Custard: the wrong thing, written the wrong way. Over the course of these articles, we've tried to investigate what software truly is.

So here's my final question :

What are you going to do now to help you write the right thing in the right way?

If you don't take something practical away with you then this series has been nothing more then entertainment . (Well, hopefully it has been entertaining). Stop and think for a moment what you are going to do to improve.

Of course this has been a very rapid, very woolly investigation into the nature of software. It's been general and vague, but hopefully it will have helped you to find ways to become a better programmer. If you really do care about your code and how you develop it, you might find my book entitled *Code Craft* an interesting read. Yes, that was a gratuitous plug. ■



Pete's book, *Code Craft*, is available in all good bookshops.

Check it out at www.nostarch.com

Taming the Lint Monster, Part 2

Anna-Jayne Metcalfe deconstructs the PC-Lint Command Line… and more.

In part 1 we introduced the PC-Lint static code analysis tool, and started to look at how to configure it. In this part we will look closely at how the PC-Lint command line is formed, and some of the common PC-Lint options. We will also consider how to configure PC-Lint for a particular project configuration, and some strategies for dealing with the analysis results it generates.

First though, let's take a closer look at the command line we closed the first part with (Listing 1).

Listing 1

```
"C:\Data\PC Lint\8.00\lint-nt.exe" -i"C:\Data\PC
Lint\8.00" -background -b --u
C:\Data\Code\Projects\Applications\SourceVersione
r\Development\SourceVersioner_vs71_Debug_Win32.ln
t -u "C:\Data\PC Lint\8.00\std_vs71.lnt" env-
vc7.lnt -t4 +ffb +linebuf -
iC:\Data\Code\Projects\Applications\SourceVersion
er\Development\Debug
c:\Data\Code\Projects\Applications\SourceVersione
r\Development\Shared\FileUtils.cpp
```

If we break this command line down to its essentials, we are left with something a bit more comprehensible:

```
lint-nt.exe -i<PC-Lint folder> -background -b --u
<project.lnt file> <std.lnt file> -u env-vc7.lnt
-t4 +ffb +linebuf -i<intermediate files folder>
<source file>
```

Put simply, the above command line defines a single file (or 'unit checkout') analysis using the configuration files `project.lnt`, `std.lnt` and `env-vc7.lnt`. It is notable that the options given in each `.lnt` file are applied in order – so those specified later in the command line can override directives contained within (for example) `std.lnt`. This is useful for message suppression purposes, as we can almost always override a high level option later in the command line.

This example also shows several PC-Lint options which are worthy of mention:

- **`-i`** specifies a folder for include and/or indirect files. In this particular case it is being used to specify the location of PC-Lint indirect files referenced within `std.lnt`, and any `.tlh` and `.tli` intermediate files located within the `Debug` subfolder under the project.
- **`-background`** instructs PC-Lint to run analysis at a low priority (useful if you are running several analysis tasks in parallel or are using a slow machine).
- **`-b`** suppresses the banner line
- **`--u`** instructs PC-Lint to ignore any files listed in the following `project.lnt` file (we will come back to this later). **`--u`** is usually used with **`-u`**.
- **`-u`** instructs PC-Lint to perform a 'unit checkout' analysis – i.e. analyse a single source file or compilation unit.
- **`-t4`** tells PC-Lint that it should use a tab size of 4 spaces while parsing source files (one of the things PC-Lint can do is check indentation).
- **`+ffb`** instructs PC-Lint that it should assume ANSI compliant for loop scoping. With compilers where this behaviour can be set by project settings (notably Visual Studio .NET 2002 onwards), the alternate option (**`ffb`**) may be required on specific projects.
- Each occurrence of **`+linebuf`** doubles the size of the PC-Lint input buffer (the default size is 620 characters).

ANNA-JAYNE METCALFE

Anna hasn't always written software for a living, but saw the light and defected to software development after several years writing 'Rusty Washer Reports' in the defence sector. She has a taste for Belgian beer (hint, hint!) and may be contacted at anna@riverblade.co.uk.

Configuring PC-Lint for specific project configurations

The file `SourceVersioner_vs71_Debug_Win32.lnt` is a project specific indirect file – an indirect file which specifies the PC-Lint options for a particular project configuration and platform. I usually refer to these as 'project.lnt files' for convenience.

A quick glance at the contents of such a file reveals that it contains preprocessor directives, the locations of additional include folders and a list of the files in the project (Listing 2).

These indirect files are critically important in ensuring that PC-Lint uses the same analysis configuration as the compiler itself. Any mismatch in preprocessor directives or include paths is likely to result in a deluge of angry (and very misleading) error messages.

It should be obvious by now that configuring PC-Lint for a given project can be a very tricky business.

If you are using Visual C++, PC-Lint can read Visual C++ project files to generate project.lnt files directly. For example, to generate a `project.lnt` file for the Unicode Debug configuration and Win32 platform of project CoreLib you could use the command line:

```
lint-nt.exe CoreLib.vcproj +d"Win32|Unicode Debug"
>CoreLib_Win32_Unicode_Debug.lnt
```

Unfortunately, current versions of PC-Lint cannot correctly handle Visual C++ projects which specify include folder or preprocessor symbol specifications using Visual Studio environment variables or inherited property sheet (**`.vsprops`**) files. If you have projects structured in this way, you will have to either write the .lnt files manually or use a third party tool to do so.

Furthermore this technique is limited to Visual C++ project files; for other compilers you will have to write the **`project.lnt`** files yourself.

Whole project analysis

The list of files shown in the example above is used in 'whole project analysis', which involves analysing all of the files in a project together. This method requires significantly more memory than analysing each source file individually, but has the advantage of allowing PC-Lint to identify issues (e.g. unreferenced functions) which only become apparent in a wider context than an individual compilation unit.

Listing 2

```
// Generated by Visual Lint 2.0.0.91 from file: SourceVersioner_vs71.vcproj
// -dConfiguration=Release|Win32
//
-D_UNICODE;UNICODE                  // CharacterSet = "1"
-DWIN32;NDEBUG;_CONSOLE             // PreprocessorDefinitions = "WIN32;NDEBUG;_CONSOLE"
-D_CPPRTTI                          // RuntimeTypeInfo = "TRUE"
-D_MT                               // RuntimeLibrary = "0"

-i"..\Include"

SourceVersioner.cpp                 // RelativePath = "SourceVersioner.cpp"
SourceVersionerImpl.cpp             // RelativePath = "SourceVersionerImpl.cpp"
stdafx.cpp                          // RelativePath = "stdafx.cpp"
Shared\FileUtils.cpp                // RelativePath = "Shared\FileUtils.cpp"
Shared\FileVersioner.cpp            // RelativePath = "Shared\FileVersioner.cpp"
Shared\PathUtils.cpp                // RelativePath = "Shared\PathUtils.cpp"
Shared\ProjectConfiguration.cpp     // RelativePath = "Shared\ProjectConfiguration.cpp"
Shared\ProjectFileReader.cpp        // RelativePath = "Shared\ProjectFileReader.cpp"
Shared\SolutionFileReader.cpp       // RelativePath = "Shared\SolutionFileReader.cpp"
Shared\SplitPath.cpp                // RelativePath = "Shared\SplitPath.cpp"
Shared\StringUtils.cpp              // RelativePath = "Shared\StringUtils.cpp"
Shared\XmlUtils.cpp                 // RelativePath = "Shared\XmlUtils.cpp"
```

If the following command line is used, PC-Lint will perform such an analysis on the entire project:

```
"C:\Data\PC Lint\8.00\lint-nt.exe" -i"C:\Data\PC
Lint\8.00" -background -b "C:\Data\PC
Lint\8.00\std_vs71.lnt"  env-vc7.lnt -t4 +ffb
+linebuf -
iC:\Data\Code\Projects\Applications\SourceVersioner
\Development\Release
C:\Data\Code\Projects\Applications\SourceVersioner\
Development\SourceVersioner_vs71_Release_Win32.lnt
```

By comparison with the previous example, the **-u** and --u options are not used and a specific source file need not be specified since the `SourceVersioner_vs71_Release_Win32.lnt` file contains a list of files to analyse.

A similar method can be used to check all files in a complete solution or workspace (much more useful, on the face of it). However, if you intend to do so please bear in mind that:

- You will have to generate a top level `solution.lnt` file which dynamically sets the preprocessor directives and additional include folders for every file in the solution – and this can easily be an order or magnitude harder than getting them right for a single project.
- The analysis runs in a single threaded process, so you will not have representative results for quite some time if the codebase is large.

In practice I find that single file analysis usually gets you 95% of the way. It is however worth running whole project analysis from time to time to identify any deadwood which has crept in.

PC-Lint messages and categories

PC-Lint organises its messages into five categories of varying severity:

- Elective Notes – e.g. 953 ('Variable could be declared as const')
- Informational – e.g. 1924 ('C-Style cast')
- Warnings – e.g 534 ('Ignoring return value of function') or 1401 ('Member symbol not initialised by constructor')
- Errors – e.g. 1083 ('1083 - Ambiguous conversion between 2nd and 3rd operands of conditional operator')
- Fatal Errors – e.g. 322 ('322 - Unable to open include file')

Conveniently, PC-Lint has a **-w** option which allows the warning level to be set globally. For example, **-w3** (the default) will enable only messages of level 'Warning' and above – so all 'Elective Notes' and 'Informational' messages will be suppressed. Similarly, **-w4** will enable all messages except 'Elective Notes'.

If you need to enable a specific message below the current warning level, you can simply add a **+e** directive for the message in question after the **-w** option. This brings us to a very useful way of detecting only specific issues – for example, if you wanted to detect any unused include files in a project, adding the options **-w0 +e766** to your command line (after all other `.lnt` files) will enable only message 766 ('Header file not used in module').

The following are examples of the sort of issues I specifically look for the first time I analyse a third party codebase (interestingly, very few of them are warnings – most are actually informational):

429	Custodial pointer 'Symbol' (Location) has not been freed or returned (Warning)
578	Declaration of symbol 'Symbol' hides symbol 'Symbol' (Warning)
716	while(1) ... (Informational)
717	do ... while(0) (Informational)
777	Testing float's for equality (Informational)
795	Conceivable division by 0 (Informational)
801	Use of goto is deprecated (Informational)
825	Control flows into case/default without -fallthrough comment (Informational)
1506	Call to virtual function 'Symbol' within a constructor or destructor (Warning)
1725	Class member 'Symbol' is a reference (Informational)
1735	Virtual function 'Symbol' has default parameter (Informational)
1773	Attempt to cast away const (or volatile) (Informational)

If you are running PC-Lint on a codebase for the first time with a (consequently) relaxed warning policy, it may just be worth turning on any issues you are specifically concerned about (using a **+e** directive in `options.lnt`) to see if any of them manifest themselves.

Common analysis failures

Some errors are invariably fatal to the analysis in nature. The ones you are most likely to see are:

- Fatal Error 314: Previously used `.lnt` file

  Each `.lnt` file can only be used once in the command line – directly or by inference. If you see this error, check your configuration files and command line for repeated `.lnt` files.
- Fatal Error 307: Can't open indirect file

Listing 2

```
Lint output file ........... : _aloa.xml
Total number of issues found : 96
Total severity score ....... : 245

File List
---------

     Rank   Score   MMsg  MSev  File

       1     81      618    3   globals.h
       2     64     1024    4   C:\progs\msvc\VC98\Include\vector
       3     24      818    2   aloa.cpp
       4     24     1510    3   parse.h
       5     20     1717    2   report.h
       6     18     1776    2   parse.cpp
       7      8     1776    2   globals.cpp
       8      6      506    3   report.cpp

Issue List
----------

     Rank  Msg    Sev   Count

       1   1776    2     20
       2    618    3     15
       3   1712    2     12
       4   1717    2     10
       5   1024    4      8
       6    118    4      8
       7   1702    2      4
       8    762    2      4
       9    783    2      3
      10   1764    2      3

Legend
------

Severity level   1 : Elective note
Severity level   2 : Informational
Severity level   3 : Warning
Severity level   4 : Syntax error
Severity level 999 : PC-lint error

Score  Severity score
MMsg   Number of issue with the highest severity encountered
MSev   Severity level of the severest issue encountered
Msg    Lint issue number
Sev    Issue severity level
Count  Total number of occurrences of issue in the whole project
```

This error indicates that PC-Lint was unable to locate a `.lnt` file on its search path. Check that the file exists in the correct location and that PC-Lint has a **`-i`** directive locating its folder.

- Fatal Error 322/Error 7: Unable to open include file

  If an include file cannot be found you will receive fatal error 322 and analysis will abort, unless error 322 has been suppressed – in which case error 7 will be raised instead and analysis will continue. In either case, check your include paths and pre-processor symbol definitions – particularly at project configuration level.

- Error 91: Line exceeds Integer characters (use +linebuf)

  PC-Lint has a fixed line buffer of 620 characters. If you see this message, add a **`+linebuf`** directive (which doubles the size of the buffer) to the command line and try again. Repeat the process until the error disappears...

- Error 303: String too long (try +macros)

  As per error 91, but this time in macro definitions. Add **`+macrobuf`** directives until it goes away.

Analysis speed

The speed of a PC-Lint analysis is influenced by many factors, of which the most significant are probably the structure of the codebase being analysed and the performance of the system the analysis is being run on. For a large codebase it is not unknown for a complete analysis run to take several hours – and needless to say, this can potentially pose problems or even act as a disincentive to run the analysis at all.

Figure 2

As PC-Lint operates directly on source files (unlike the .NET assembly analysis tool FxCop, for example), it must read and process include files; therefore the structure of include dependencies in your project can be very significant in terms of analysis time. Reduce your include dependencies (which you already do routinely, I assume…?) and the analysis time is likely to be reduced. Similarly, a reasonably specified modern system (e.g. a Core 2 or Athlon X2 machine with 7200rpm disks and 2GB or more memory) will crunch through your codebase in a far shorter time than that cruddy old P4 box that was co-opted as a build server 4 years ago.

As current PC-Lint versions are entirely single threaded, adding more CPU cores will not reduce the analysis time unless you somehow run multiple analysis tasks simultaneously. On multicore systems it is entirely practical to do this (single file analysis is particularly amenable to parallelisation), and it can reduce analysis times quite significantly in some cases.

One interesting recent development is that PC-lint 9.0 (released last Autumn) adds options for precompiled and bypass header support - which allow the contents of system include files to be preprocessed and cached in a similar way to the Visual C++ precompiled header compilation mechanism. The tests I have run suggest that these options can potentially cut analysis times by a factor of at least 3 to 4. Unfortunately, the current version of PC-Lint (9.00b) seems to generate extraneous errors when precompiled headers are activated (I was analysing a Visual C++ project, so that could be a result of something specific to the Visual C++ system header files, of course). No doubt any such issues will be resolved in a future update to the software, at which point these options could become very useful.

Overall, although techniques such as parallelisation and precompiled headers can significantly reduce your analysis time, I would strongly recommend that you take a close look at the structure of your codebase and the performance of the system the analysis is being run on first.

Tuning out issues in libraries

Regardless of which libraries you use in your projects, you are likely to encounter PC-Lint issues in class or macro definitions within those libraries. When you don't have any control over the implementation of the libraries concerned, such 'noise' can be irritating to say the least (athough it could of course be indicative of a real problem with the library implementation you should be aware of…).

Fortunately, using some of PC-Lint's error suppression directives it is in most cases relatively easy to write lint directives to prevent this happening, for example:

```
-emacro(1924, MAKE_HRESULT)
-esym(1932, ATL::CAtlExeModuleT<*>)
-etype(1746, boost::shared_ptr<*>)
```

The directives **-emacro**, **-esym** and **-etype** suppress issues in macros, named symbols, and instances of objects or values of the given type respectively (note the availability of wildcards here). PC-Lint has quite a number of such options for fine-grained control of warning policy, and it is well worth getting to know how to use them effectively. By the way, you can also use them in the code directly by prefixing them with a //lint comment, for example:

```
//lint -esym(1712, CoreLib::CSomeClass)
```

A special mention should be reserved for the 'big hammer of last resort' for noisy library header files. It looks something like this:

```
//lint -save -w0
#include "NoisyHeader.h"
//lint -restore
```

The meaning of the above should be fairly self-explanatory – the header in question has so much wrong with it that we're just going to disable any issues from it for now. Of course, if you find yourself doing this as anything but a short term measure you really should be asking whether the header in question code has a long term place in your codebase...

Incidentally, over the past few years I have amassed significant collection of such 'tuning' directives for issues within the Win32 libraries (Win32 API, ATL, WTL and MFC) which have been collected into specific indirect files so we don't have to specify them directly in each project or in our global warning policy. You can download them from [2] if needed.

Turning down the volume – How to cope with a deluge of analysis results

I am pretty sure that most developers who run PC-Lint do so using either pre-configured scripts or some form of rudimentary integration into whichever IDE they are using.

Regardless of the method used, the issue of how to deal with the volume of analysis results it can produce remains.

Common strategies include turning off all but the most critical issues (with the intention of gradually enabling others as the issues are addressed š not that that always happens of course), grepping the raw analysis results, exporting to a spreadsheet and filtering there....the list goes on.

Regardless of the method used, the issue of how to deal with the volume of analysis results it can produce remains

Which method you use really doesn't matter. What matters is that you find an approach which works well for you and your organisation, and continue to refine it as appropriate (the same applies to your warning policy, of course).

(free) Tools which may help

While these techniques can be very efficient ways to spot specific issues on your 'watch list', they may not give you much of a feel for the state a codebase as a whole is in. For that, a tool such as Ralph Holly's Aloa [3] or LintProject [4] can be helpful.

Aloa ('A Lint Output Analyser') takes advantage of the fact that PC-Lint can be configured to produce XML output, and interprets the results to produce a text based summary of the issues encountered (Listing 2).

If you are looking for a way to get to grips with an unfamiliar codebase, Aloa is well worth checking out.

LintProject (a tool I wrote several years ago and released as freeware via codeproject.com) takes a different approach: it runs a simple unit checkout analysis on complete Visual C++ solutions and produces simple HTML reports of the results, organised by project and file.

I wasn't aware of Aloa at the time, and we needed a simple way to analyse the codebase of our main product – 80 projects comprising about 250,000 lines of code. Obviously, a manual per file or per project analysis was not going to work, and we needed a way to analyse a complete solution in one hit. Since PC-Lint does not provide a way to do this 'out of the box' (and neither does Aloa, unfortunately) that meant writing something new.

Analysis tools such as PC-Lint can uncover real problems in your codebase

Like Aloa, LintProject is a simple command line tool which runs PC-Lint and captures its output. Unlike Aloa, LintProject runs on complete Visual C++ solutions or workspaces (whichever terminology takes your fancy) and produces simple HTML reports describing the number of issues found in each project and file. Although the output it produces is far less detailed than Aloa, it is a useful way to get a feel for the analysis state of a complete Visual C++ solution.

Conclusion

Analysis tools such as PC-Lint can uncover real problems in your codebase. Unfortunately, far too many developers are either blissfully aware of the capabilities of code analysis tools or sceptical of their usefulness (sadly many even see compiler warnings as unwanted 'noise and would rather turn down the warning level rather than fix the root causes).

As a result advocating the use of such tools on a project can be a bit of an uphill struggle – unless the project is failing, in which case they may well grasp at anything for a while.

However, in our experience it is definitely worth investing the time to integrate the use of such a tool into your development process. Although you can expect to be warned of significant numbers of issues in your code when you first analyse it, if you persevere the analysis tool is likely to repay the investment you put into it many times over by assisting you in increasing the quality of your code.

Like any tool, if you take the time to learn how to use it effectively you will be more likely to reap the best results. ■

References

[1] http://www.gimpel.com
[2] http://www.riverblade.co.uk/products/visual_lint/downloads/PcLintConfigFiles.zip
[3] http://www.ddj.com/cpp/184401810
[4] http://www.codeproject.com/KB/applications/lintproject.aspx

xCover: Code Coverage for C/C++

Matthew Wilson looks under the bonnet of the xCover open-source library.

This article describes a novel use of a non-standard pre-processor feature available with three leading C/C++ compiler collections to provide limited but effective automated code coverage for C and C++, in the form of the xCover open-source library.

Introduction

Ordinarily, I like to write articles that describe some invention that is safer and/or faster and/or more flexible that the current state of the art, such as those in last and next months' instalments of Overload about a certain formatting library. This is not one of those articles.

The library discussed, xCover, is one of three simple, indeed humble, libraries that allow me to apply software quality assurance measures to my open-source and commercial work. xCover is a code coverage library. The other two, xContract and xTests, are software contract enforcement and automated (unit & component) testing libraries, respectively. Aside from correctness, a *sine qua non* of all quality software, the primary design characteristics of these libraries are:

- modularity, so they can be bundled with my open-source libraries, and
- portability, so they work with the range of compilers, architectures and operating systems with which the open-source libraries work

Naturally, flexibility, expressiveness, discoverability and transparency, and efficiency are all nice to have as well, and they have been accounted for where possible, but they nonetheless play second fiddle. Thus, each library has competitors in their respective areas against which they may lose when compared in such terms.

Now that I've totally undersold you on xCover, where's your motivation for reading on? Well, for one thing, the manner by which code coverage is achieved for C and C++ by pure library measures involves an interesting, nay devious, twist on a (non-standard) pre-processor facility. Furthermore, being a library, and an open-source one at that, it means code-coverage is available to people don't wish to spend money on proprietary commercial tools, or who are not using development suites that include code coverage tools (such as GCC's gcov).

Code coverage

We don't have the time, and I don't have the expertise, to give a full discussion of code coverage here. Instead, I'll stipulate that coverage can take account of the **where**, **when**, **why** and **how much** of execution. Consider the function `process_int()` in listing 1.

The simplest aspect of code coverage analysis, **where**, consists of being able to detect if execution passes through the lines 4, 9 and 13. **when** and **how much** consist of recording each time execution passes through these lines. Most complex, **why**, would involve determining which part of the compound expression on lines 5–7 is active: is it that value is positive, or is it that value is negative and the Boolean control parameter `ignoreNeg` is non-zero. (Note: I wouldn't recommend this coding style. It's just for pedagogical purposes.)

Listing 1

```
1   void fn1(int i) {}
2   void fn2(int i) {}
3   void process_int(int value, int ignoreNeg)
4   {
5     if( value > 0 ||
6         ( ignoreNeg &&
7           ((value = -value), value > 0)))
8     {
9       fn1(value);
10    }
11    else if(value < 0)
12    {
13      fn2(value);
14    }
15  }
```

xCover primarily supports **where** analysis. Currently it does not record **when**, and it records but does not report **how much**. It does not record **why**, and I cannot foresee it being able to do so. As such it is, therefore, only a partial code coverage solution, although in practice it seems to be a useful portion.

__COUNTER__

Since version 7.0 of Visual C++, version 8.0 of Intel C/C++ and version 4.3 of GCC, these compilers support the non-standard `__COUNTER__` pre-processor symbol, which resolves to an integer whose value starts at 0 and increments by 1 each time its value is used. Consider the program in Listing 2.

This produces the output:

```
__COUNTER__: 0
__COUNTER__: 1
__COUNTER__: 1
```

Note that testing for its existence does not increment the counter, only using its value.

Where-coverage workflow

The basic workflow of an xCover-ed component is relatively simple. The programmer marks one or more lines within each block with the `XCOVER_MARK_LINE()` macro, which is implemented in terms of

Listing 2

```
#include <stdio.h>

#define GET_COUNTER()   __COUNTER__

int main(int argc, char** argv)
{
  printf("__COUNTER__: %d\n", __COUNTER__);
  printf("__COUNTER__: %d\n", GET_COUNTER());
#ifdef __COUNTER__
  printf("__COUNTER__: %d\n", GET_COUNTER());
#endif
  return 0;
}
```

MATTHEW WILSON

Matthew is a software development consultant,who belives strongly that C++ software must be robust, efficient and expressive: if not, you should be using something else. Matthew is currently working on *Breaking Up The Monolith: Advanced C++ Design Without Compromise C.*

`__COUNTER__`, along with other standard stuff, such as **`__FILE__`**, **`__LINE__`** and **`__FUNCTION__`**. The code is then compiled and linked with xCover's object library. (Note: C applications must call **`xcover_init()`** and **`xcover_uninit()`**, to ensure the library is initialised. C++ applications need not do so, as this is implicitly handled by Schwarz counters [1], [2].)

During execution, as each marked line is executed, the xCover API function **`xcover_markLine()`** is called, passing in the file, line, function and counter.

```
xcover_rc_t
xcover_markLine(
  char const* filename
, int line
, char const* function
, int counter
);
```

If the file is not already known to the library, a file record will be created and added to the file map, where it is associated with a vector of mark records. The vector indexes correspond to the counter values. If the counter does not already contain a mark record for the counter value, it is expanded accordingly, and the mark record is initialised with the file, line and function. The use-count of the mark record is increased.

At any time the user can request a report, and any mark records in the vector (for the file corresponding to the report request) that have a use-count of zero will be reported as uncovered code.

Let's put our **`process_int()`** function into an xCover program (Listing 3) so you can see how this works.

There are several things to note:

- The use of **`XCOVER_MARK_LINE()`** at lines 8, 13, 18 and 21 fully mark the possible execution paths in **`process_int()`**.
- The explicit initialisation of the xCover library (lines 25 and 36), because it's a C program
- The (incomplete) exercising of **`process_int()`** in lines 33 and 34.
- The reporting of the coverage from the current file, in line 35

When executed, this produces the following output:

```
[Start of file ../cvu.ex.2.c]:
Uncovered code at index 2 in file ../cvu.ex.2.c,
between lines 13 and 21
[End of file ../cvu.ex.2.c]: 1 uncovered block(s)
```

Files, aliases, groups

Naturally, most of the time you're not interested in code coverage of a single file containing **`main()`**. In order to support this, the library provides three ways of grouping sets of code coverage marks: by file, by file alias and by group.

Many components, particularly header-only (including template) ones, come in a single file. As such, it is appropriate to be able to request coverage reports on a file. This is done via the xCover macro **`XCOVER_REPORT_FILE_COVERAGE(filePath, reporter)`**. Users may specify the full file path, or just the file name, or file-name and extension, or a pattern. (Pattern matching can be done with UNIX's **`fnmatch()`**, Windows' shell's **`PathMatchSpec()`**, or another of my open-source libraries, shwild.) One reason for this is that different compilers define **`__FILE__`** differently: it can be absolute or relative; if you're on Windows it might contain forward slashes, or backwards slashes, or a mix of the two. For example, you might see any of the following forms, depending on your compiler:

```
../../include
H:/freelibs/fastformat/0.3/include
H:\freelibs\fastformat\0.3/include
```

Note that if your search criteria are too general, you will receive reports for all matching files. Here are some examples from my code base:

Listing 3

```
1   /* file: cvu.ex.2.c */
2   #include <xcover/xcover.h>
3   #include <stdlib.h>
4   void fn1(int i) {}
5   void fn2(int i) {}
6   void process_int(int value, int ignoreNeg)
7   {
8     XCOVER_MARK_LINE();
9     if( value > 0 ||
10        ( ignoreNeg &&
11          ((value = -value), value > 0)))
12    {
13      XCOVER_MARK_LINE();
14      fn1(value);
15    }
16    else if(value < 0)
17    {
18      XCOVER_MARK_LINE();
19      fn2(value);
20    }
21    XCOVER_MARK_LINE();
22  }
23  int main(int argc, char** argv)
24  {
25    int r = xcover_init();
26    if(r < 0)
27    {
28      fprintf(stderr, "could not initialise
          xCover library: %s\n",
          xcover_getApiCodeString(r));
29      return EXIT_FAILURE;
30    }
31    else
32    {
33      process_int(0, 0);
34      process_int(-1, 1);
35      XCOVER_REPORT_THIS_FILE_COVERAGE(NULL);
36      xcover_uninit();
37      return EXIT_SUCCESS;
38    }
39  }
```

```
XCOVER_REPORT_FILE_COVERAGE("*stlsoft/*/
string_to_integer.hpp", NULL);
XCOVER_REPORT_FILE_COVERAGE("simple_string.hpp",
NULL);
```

The second parameter is for a custom reporter, which is not yet fully supported. If you specify **`NULL`**, you get the default reporter, which writes to **`stdout`**.

As well as reporting by file name, it's also possible to use aliases and groups.

An alias is simply a user-defined alternate name for a file, defined within the file (in a place that will be executed) by the macro **`XCOVER_CREATE_FILE_ALIAS(aliasName)`**. Aliases were my original solution to the problem of different file path representation by different compilers, before I added the wildcard pattern matching. But they can still be useful if, say, you use different files to implement a component depending on operating system, threading model, and so forth. Groups enable you to associate multiple (related) source files into one logical unit, via the macro:

```
XCOVER_ASSOCIATE_FILE_WITH_GROUP(groupName)
```

Again, this is done within each associated file in a place that is guaranteed to be executed. (Yes, I recognise the circularity in having a code coverage library rely on certain parts of the code being executed, but in practice it works well enough.)

Drawbacks

Gaps

The most obvious drawback of the library is the issue of gaps. If, say, your source file contains ten **XCOVER_MARK_LINE()** statements, but neither of the first two are executed, you will get a message reporting along the following lines:

```
Uncovered code at index 1 in file abc.cpp, between
the start of the file and line 34
```

Although somewhat lacking, it still gives useful information, and it is sufficient to track down your uncovered code. If, like me, you tend to have highly structured files, with extensive header information, includes, compiler compatibility pre-processor testing, and so forth, it can actually be many tens, or even a couple of hundred lines before you get to your first line of executable code, which can mean that finding the first (unexecuted) mark can be a pain. In order to work around this, you can use the macro **XCOVER_MARK_FILE_START()** to designate the start of your file. You can also use this to skip helper functions at the head of the file that whose coverage you don't care about. The corresponding macro **XCOVER_MARK_FILE_END()** allows you to achieve the same results at the end of file.

It's more important to use **XCOVER_MARK_FILE_END()**, and in a place that's guaranteed to be executed, because if you miss the last two of ten you may never know about it.

Currently, a big problem is when *none* of the statements within a file are executed. In that case the current version of the library does not even know about the file. So, rather than getting a report that all of the statements from your file are missing, you get no report at all, and everything looks hunky dory.

In practice, this has proven much less problematic than it seems, since I almost exclusively use xCover in association with automated testing. When used to determine runtime coverage within applications it would be more bothersome, but this could be reasonably easily fixed by using Schwarz counters to ensure each compilation unit tells xCover about its presence. I plan to address this in a forthcoming release.

Conflict with other uses of __COUNTER__

This is an issue I've not yet encountered since I've never used, nor even come across, any code that uses **__COUNTER__**. (I have mused as to whether it's one of those architecture astronaut features, that someone thought might have a good use but there's never been a convincing use-case. Or maybe I'm just missing something. Whatever the case, xCover is the unwitting beneficiary.)

Anyway, if you do use **__COUNTER__** for other things, then clearly xCover is going to be at risk of producing false positives. A feature planned for the near future will be the ability to specify a **STOP** and **START** pair of macros that could be used around the other use(s) of **__COUNTER__**. Naturally, that's going to be quite visually intrusive in your code. Which brings us to another issue.

Visual pollution

Having to have **XCOVER_MARK_LINE()** at every branch in your code leads to a lot of stuff filthying up your beautifully crafted code, and can detract from transparency. As yet I have no solution to ameliorate this effect.

Of course, all well-crafted software will have contract enforcement code providing, at least, pre-condition enforcements. It'll likely already have function-entry logging statements – controllable at runtime, of course – allowing you to track the behaviour of your software at any point in its lifecycle. So you'll have a fair amount of such stuff already. Personally, my brain has adjusted to these kinds of intrusions into functions/methods, and I believe that if you structure your code it's reasonably easy to navigate your way past much of this function-entry stuff.

However, when you're considering marking every block, there's no denying that it's intrusive. Currently, there's no answer to this. Although there are some plans, as described in the next issue.

Auto-insertion

So far, I've survived with keyboard macros and search-replace to insert all the **XCOVER_MARK_LINE()** (or equivalent) statements into code. However, I do have plans to write scripts to do this (as I have to insert Pantheios logging statements throughout the codebases of my clients). These would find every occurrence of the opening curly brace and replace it with the opening curly brace + **XCOVER_MARK_LINE()** (or equivalent), except:

- when there's already one there, or
- when it's being used to define a namespace, enum, struct, class or union, or
- when it's in a string or character constant, or
- any other exceptions I've not yet thought of

Further to this, I've considered writing a pre-pre-processing filter that would read in your source file and insert these automatically, before passing to the pre-processor. That way, you wouldn't have to have all this evil looking junk in your beautifully succinct code. (I'm more than happy for any gentle readers to take this idea on, and make it their own.)

Missing curly braces

A problem with auto-insertion occurs if you don't always use curly braces, as in:

```
if (x < y)
  y = x;
```

I don't wish to be unduly offensive, but if you write code like this, then you've got much bigger problems than code coverage. Never, ever, ever, write conditional code without curly braces. Ever.

So, given that haughty imperative, we don't have any problems. Er, except for the tertiary operator. Once again, this is a part of the problem space that, like compound expressions – remember the **why** problem – xCover cannot handle. It is only a library, after all.

What to do with unsupported compilers

For the moment, there's nothing to do for unsupported compilers, other than ensure that your source code works in either case. Listing 4 is an extract from the code coverage header file in one of my client's codebases, with names changed to protect the guilty.

So, all the C and C++ code in company XYZ's codebase uses **XYZ_MARK_LINE()**, which resolves to nothing if the compiler doesn't support **__COUNTER__** (and **__FUNCTION__**) or the particular build configuration (e.g. release) requires coverage testing to be suppressed.

Alternatively, a pre-pre-processor that inserts the mark line statements could provide the counts directly into the code for those compilers that don't support **__COUNTER__**, simultaneously removing any possible conflicts with other uses of **__COUNTER__** for those that do.

Code coverage for (open-source) libraries

As any of you know who use them know, the movement from one STLSoft version to the next takes place at glacial pace in part because I am still (by accident more than design) a one-man band, and mainly because I'm always coming up with new libraries that are more interesting to develop. One of the things that's holding me up with the move to STLSoft 1.10 is the need to sort the wheat – there's some good stuff in there – from the chaff – there's also a lot of experimental, unused, oxygen-thieving junk in there – and also my desire to get full code coverage in all the libraries. As part of this millennial effort, I've been working through the components and adding code coverage. If you want a sneak peek, you can download the STLSoft 1.10 alpha distributions.

Listing 4

```
/* /////////////////////////////////
 * File: xyz/quality/coverage.h
 */

...

/* /////////////////////////////////
 * Includes - 1
 */

#include <xyz/xyz.h>
#include <stlsoft/stlsoft.h>

/* /////////////////////////////////
 * Compatibility - 1
 */

#ifndef XYZ_QUALITY_NO_USE_XCOVER
# if defined( \
     STLSOFT_PPF_COUNTER_SYMBOL_SUPPORT) && \
     defined(STLSOFT_PPF_FUNCTION_SYMBOL_SUPPORT)
#  define XYZ_QUALITY_USE_XCOVER
# endif
#endif

/* /////////////////////////////////
 * Includes - 2
 */

#ifdef XYZ_QUALITY_USE_XCOVER
# include <xcover/xcover.h>
#endif

/* /////////////////////////////////
 * Compatibility
 */

#ifdef XYZ_QUALITY_USE_XCOVER
# define XYZ_MARK_LINE()
XCOVER_MARK_LINE()
#else
# define XYZ__MARK_LINE()
stlsoft_static_cast(void, 0)
#endif
```

What's been somewhat surprising to me in this process is just how many defects have lain dormant in code that I'd believed correct. Take STLSoft's **`basic_simple_string`**, for example. Putting in code coverage helped me find one branch of dead code, and four serious defects in what must have been previously untravelled code paths!

Given this, I am planning to incorporate xCover into most/all of my open-source projects, including bundling it with them and incorporating it into the automated tests. The recently released FastFormat library contains code coverage statements throughout its core, and I plan to ensure that the Pantheios and recls libraries get the same treatment in the coming months. I wait with some trepidation to see what might come out of this.

Code coverage for application code

When dealing with my own libraries, polluting the source with coverage mark statements is defensible because (i) I own them, (ii) they must support a lot of different applications in a lot of different ways, and (iii) they are used a lot more than they are changed. While the first two arguments are equally applied to proprietary application code, in practice the third may not be. In consequence, the transparency of the code – how easy it is to understand it in order to change it – achieves a higher significance than it does for libraries.

In this case, it may be reasonable to restrict the application of the code coverage macros to each function entry, and then to those other branches that are deemed important. It's really down to best judgement to balance the advantages of full coverage with the reduction in transparency.

Another factor to bear in mind is that application code usually has to exist on fewer compiler/architecture/operating-system permutations, and it's more likely that a compiler-related tool (e.g. GCC's gcov) or a commercial product is justifiable.

Summary

This article has introduced the xCover open-source C/C++ code coverage library, based on the non-standard **`__COUNTER__`** pre-processor symbol. It has illustrated how where code coverage can be achieved via insertion of **`XCOVER_MARK_LINE()`** statements that record the execution path in the xCover library, and how gaps in the execution path may subsequently be reported. The numerous drawbacks and limitations of the library have been discussed, and measures used to ameliorate and/or obviate them have been highlighted, where available.

The library does not attempt to be a universal solution to the code coverage problem, but it has already proven to be of considerable use in open-source and commercial developments as a measure for improving code quality.

The library is still in the early stages, and feedback and assistance will be equally warmly received. ■

References

[1] *C++ Gems: Programming Pearls from the C++ Report*, Stanley Lippman (editor), Cambridge University Press, 1998

[2] *Imperfect C++: Practical Solutions for Real-Life Programming*, Matthew Wilson, Addison-Wesley 2004

Mailbox @ C Vu

Your letters and opinions.

Code Critique 54 had many good entries. Let me add some comments, especially to Roger's winner statement:

> I was pleased to see Ken's entry this time round, where he was able to find the two biggest problems with the code by simply (!) running PC-Lint. While being able to read code and find faults is a fundamental skill, having the ability to use a tool to perform the same task enables much more reliable coverage of complete code bases. So this time I have decided to award the prize to PC-Lint for finding the problems; Ken gets the prize for using the right tool for the job!

To avoid misunderstanding, let me state up front, that Ken did a great job and deserves the price. My problem is with the 'rationale' statement. I'm also not against using Lint, even embedded in the process, just IMO it did a lousy job in the particular case. And the right tool for the case is still the expert code reviewer, who received a questionable amount of help.

Let's take a close look at what Lint flagged and the relation to the case.

1. Ignoring return from **`strftime`**. I agree with Ken that in general it's bad to ignore return values, especially those reporting errors. But there are functions where we really don't care about the return. A frequent question on forums is what (void) prefix does before a function, and answers are minority: to explicitly state we ignore the return, majority: this is old-time uglyfication to shut up lint. E.g. before **`strcpy`**.

   Ken suggests the function could return 0 indicating indeterminate buffer, and that it would be good to assert. In general, I am a big fan and promoter of asserting, but this case is also an exception. The buffer here is fixed at 20, the format string is a literal, and the standard states the behavior: our string can be at most 2+5+2+2+2 chars in variable 5 bytes in fixed, that is overall 18. So in this state assert only would check our standard lib works. We normally do not spend code on that. Assert could defend against later modification of the format string without also adjusting the buffer – I would not accept it for a local buffer that is used for this and only this purpose, and in a change-reviewed environment.

   Nevin addressed this by defining buffer in a cute way: **`char buffer[sizeof("31 Jan 12:34:56")];`** too bad it is not correct in content, the month field should be 5 characters.

   The defence I use in practice is to avoid **`strftime`**, I have a suite of functions **`str_printf`**, **`str_strftime`**, that take all the same input and return **`std::string`**. The implementation covers all the cases and figures out the length. At the cost of a few cycles eliminating all the related problems for good. and saving the client side from both length-related checks, and guessing. But if one is not using a like approach and is determined to use **`strftime`**, it is fit for the purpose here and is okay. Lint flagged a false positive. And gets an extra bad mark, because it could theoretically figure out the needed and passed length.

2. Expression always **`false`**. Yeah, indeed. Though, it reflects on the problem that everyone noticed at first glance, just seeing plain **`enum`** and being used as mask. Also, suppose the **`enum`** was a little different, inserting a fourth level **`INFO`** in the front, and the rest of the code kept as is. (Meaning no macros or whatever introduced for it yet). The code is as broken as the original, but Lint is now silent. I'd say it flagged the bug by a lucky accident.

3. oss hiding other oss. Roger classified it as a hit, but let's look in the details. The hiding of outer oss is intentional and good. The macro uses **`{}`** for that very reason. It wants the hiding. In C++ hiding is natural and made in by design. In my experience flagging it is counter-productive, and just creates noise. One of my recent project was even hit by a motion to turn on this warning and 'clean up' code – the result was at least one bug introduced to correct code and several places of uglyfication.

   The bug in the code is not with hiding, but that the macro parameter **`X`** is used incorrectly. In the test case it then hurts by using the wrong oss. The correct macro would have an extra line as first thing inside **`if`**: **`const std::string& msg = (X);`** And use **`msg`** in the streaming line where **`(X)`** was. Then the code would work fine, and do what was expected. And Lint would still report that there is hiding.

4. **`main`** should return a value. Though I also tend to write explicit return 0, I flag it as a bug in Lint. Probably it has a simple checker that reports missing returns from non-void functions. Not having the exception list with **`main()`**. A tool should be sharp and tell problems from non-problems, here it fails.

   I agree with Ken, that if you use a static checker you better deal with flagged issues first – for the case I'd consider the net usefulness.

And we have still not mentioned the main issue that made me write in the first place. Which is the example's most dangerous problem? In my evaluation **`printf(userstring)`** wins, hands down. Why? Because the rest is easily found by a little testing of the module. Even if not, it makes the log system somewhat lame. But the effect of the **`printf`** format-string vulnerability can easily make it to production and cause disaster. Just have **`"% n"`** somewhere in the log string, and you get some random 4 bytes patched in your memory. With luck hitting a non-paged address, causing crash, in other cases just modify some data element in a subtle way. Try to think possible consequences. Or the hours of headache until one is able to track back the problem to the auxiliary logging system....

Did Lint flag this? No. Could it? Easily. Compilers can flag it by themselves too, just turn on the proper options. (Also most compilers have a warning for **`if`**-expression being constant; ICC also has a remark for hiding...)

In summary the only prize I'd offer Lint here is the raspberry.

Balog Pal
(pasa@lib.hu)

Editor's note

Discussion about the code critiques is encouraged in the 'Dialogue' part of CVu. What do you think – do you agree with these comments or would you like to defend Lint? Email the editor for inclusion in the next CVu or post directly to accu-general@accu.org!

If you read something in C Vu that you particularly enjoyed, you disagreed with or that has just made you think, why not put pen to paper (or finger to keyboard) and tell us about it?

Code Critique Competition 56

Set and collated by Roger Orr.

Please note that participation in this competition is open to all members, whether novice or expert. A book prize is awarded for the best entry. Readers are also encouraged to comment on published entries, and to supply their own possible code samples for the competition (in any common programming language) to scc@accu.org.

Last issue's code

I'm trying to write a C++ SQL framework that uses the **operator<<** idiom to add bind variables. I'm having problems getting the operators right – I've stripped it down to the following code that won't compile. Using MSVC I get a complaint about **bind** on the last line of **main**; with g++ the **Insert** and **Query** examples won't compile either. Can someone help me sort this out?

Listing 1

```
// ---- db.h -----
#include <string>
#include <vector>

namespace DB
{
  class Row
  {
  public:
    std::string toString() const;
    // ... other method elided
  };

  static struct exec_t
  {
  } execute;

  class SqlBase
  {
  public:
    void bind( int & );
    void bind( std::string & );
    // ... other methods elided
  };

  class Query : public SqlBase
  {
  public:
    Query( std::string const & query );
    std::vector<Row> operator<<( exec_t & );
  };

  class Insert : public SqlBase
  {
  public:
    Insert( std::string const & statement );
    void operator<<( exec_t & );
  };

  // Use template to retain type of 'stream'
  template <typename T, typename U>
  T & operator<<( T & stream, U & value )
  {
    stream.bind( value );
    return stream;
  }
}
```

Listing 1 (cont'd)

```
// ---- test.cpp -----
#include <fstream>
#include <iostream>
#include <string>
#include <vector>

#include "db.h"

int main()
{
  using namespace std;
  using namespace DB;

  int id;
  string name;
  string filename( "output.txt" );

  // populate id and name ...
  Insert( "insert into employee(name,id)"
    " values(%,%)" )
    << name << id << execute;

  vector<Row> result = Query( "select name "
    "from employee where id = %" )
    << id << execute;
  string firstrow = result[0].toString();
  ofstream( filename.c_str() ) << firstrow;
}
```

Critiques

Balog Pal <pasa@lib.hu>

Wow, now this is a fine CC! Let's start with the reported problems. VC states this:

```
scc55.cpp(52) : error C2039: 'bind' : is not a
member of 'std::basic_ofstream<_Elem,_Traits>'
  with
  [
    _Elem=char,
    _Traits=std::char_traits<char>
  ]
scc55.cpp(85) : see reference to function template
instantiation 'T &DB::operator
<<<std::basic_ofstream<_Elem,_Traits>,std::string>(
T &,U &)' being compiled
  with
  [
    T=std::basic_ofstream<char,
      std::char_traits<char>>,
      _Elem=char,
      _Traits=std::char_traits<char>,
      U=std::string
  ]
```

ROGER ORR

Roger has been programming for over 20 years, most recently in C++ and Java for various investment banks in Canary Wharf and the City. He joined ACCU in 1999 and the BSI C++ panel in 2002.
He may be contacted at rogero@howzatt.demon.co.uk

It is quite informative, the compiler wants to use `<<` defined in DB. And fails with template instantiation due to call `bind()` on `ofstream`. This should make us realize that

1. a templated `operator <<` picking up almost anything is not a good idea, if it is really a special purpose thing.
2. `using` directive for DB was probably meant only for a few types, but imports everything, including this template – and it takes over those in `std::`.
3. most `std::<<` operators work with `ostream`.
   `cout << firstrow` would work, but we're passing `ofstream`, and it is better fit for template than even a simple to-base conversion!

More on this later, let's look for the other reported errors. Having no gcc around, I tried Comeau online with the code, and it reports

```
"ComeauTest.c", line 72: error: no operator "<<"
matches these operands
operand types are: DB::Insert << std::string
<< name << id << execute;
^

"ComeauTest.c", line 76: error: no operator "<<"
matches these operands
operand types are: DB::Query << int
<< id << execute;
^
```

We can think DUH, we have that omnipotent `<<` and the compiler says no match? What's going on? Is it not seen?

If we fiddle a little with the code, and make `Insert` a named object, then use the rest of the line, it gets accepted. The difference should be a clue: `Insert(...)` and `Query(...)` is an rvalue, and the template wants to bind it to a `&`, not `const&`, and this is not allowed. If we make the template taking `(T const & stream, U& value)`, it gets selected for the original line.

This behaviour surprises me; if asked, I would guess deduction could chose `'T=const Insert'` to meet a `T&` signature (as for references, the standard allows using a more cv-qualified type at 14.8.2.1p3). Apparently this does not apply to the rvalue case, it is simply not fit for `T&` place, period.

Stepping back to VC: why does it accept this, is it an extension? Looking at the options we find (at properties/ C/C++ / Language), that extensions are enabled by default, turning it off – same as `/Za` switch – bingo, we have the same errors here too. A relief to have compilers that agree.

(For a moment at least... I tried some more formulas with rvalues, and discovered a difference. On the right hand side `<< name` works fine. `<< (string) "a"` fails on both. However, `<< (const string) "a"` is accepted for the template by Comeau (`U = const string`), not so by VC. While passing a named `const` string is accepted by VC too. Templates are still like black magic...)

Now we know the immediate problems, what can be done? Patching up VC with extensions enabled was not so hard – I just removed the `using namespace DB;` and inserted using declarations instead: `using DB::Insert; using DB::Query; using DB::Row; using DB::execute;` Alternatively, we could use those prefixes directly in the code.

It is worth thinking about why it works, how the compiler selects the correct stuff. The trick is called ADL, short for argument dependent lookup, aka Koenig lookup. It says that if there is a function or operator call, where arguments are a type from some namespace, the content of all those namespaces are included in the overload selection. Thus having `DB::Insert` on left of `<<` makes `DB::<<` considered. Similarly having `std::ofstream` or `std::string` there makes all the `<<` operators in `std::` considered. (so `using namespace std` can also replaced by a few using declarations to have `string`, `vector`, and some frequently used names usable without prefix, even having them in a dedicated header for the company; and leaving the zillion rarely encountered names hidden, used with prefix).

Too bad this easy patch doesn't work with turning standard mode back on (and for the other compilers without relaxed `&` binding). While non-const member functions can be invoked on a rvalue, for nonmember functions rvalues bind only to `const&`. The `ofstream` call uses `<< operator` in `std::`, so we're out of luck. Possibly we could create some more overloads in a visible place, but then face duplication, side effects, so the cure seems worse than the disease. Instead just go the normal way, use named `ofstream`, then use `<<` on that. The original line is fishy anyway: there is no chance to know whether the operation succeeded or not. Normally for 'real' i/o streams we want to check errors and report errors at some place. (After explicit `close` too.)

The `DB::` elements are different, they are not really streams, and can be made to report errors in a sensible way, say throwing exception – and the single line usage shown in `main` makes sense after all. (Until getting to this point I had 'who on earth would want code like that' in my head, but the usage made the 'acceptable IF' category ;-), especially if the application is built around these operations.)

So, as the problem is, as discussed, the nonmember `<<` taking `&`. The first idea could be a blatant cheat:

```
template <typename T, typename U>
T & operator<<( const T & stream_c, U & value )
{
  T& stream = const_cast<T&>(stream_c);
  stream.bind( value );
  return stream;
}
```

Having this template, the code works, and does what the original intended. Unfortunately it picks up some cases it should not compile: if someone has a `const Insert` object, or a function was passed a `const&` with intention not to tweak. Without the `const_cast` the attempt would be caught by the compiler. Now it must be caught by review. If the intent is as suggested, there is no real need to have `const` versions of these objects, or pass them around – so it can work. However if they are normally used as class members or passed around between functions, it is a risk.

Seeking a better solution, we notice that expression `Insert( "" ) << execute;` works fine. As mentioned earlier you can call member functions on rvalue, including non-`const` ones, and this version is a member. How about getting rid of the freestanding template, and using a member instead? Add this to `Insert`:

```
template <typename U>
Insert & operator<<( U & value )
{
  this->bind( value );
  return *this;
}
```

And the same into `Query`, that returns `Query`. Try: it works fine.

No more offending `const_cast`, though we have some duplication. We could try to put it to `SqlBase`, but then the problem of losing the type info comes back. And to handle it a very similar template must be re-introduced in `Insert`/`Query`. Or go the other way around, have all `<<` in `SqlBase`, then issue a virtual call from the one that processes execute. But those functions have a different return type.

A third approach, which actually works, could use a different approach, sacrificing the uniform execute. If instead we had `executeI` and `executeQ`, it ties the correct procedure and return type. Creating a possibility to mismatch: `Query(...) << executeI;` but that can be caught at compile or link time. This way we can even eliminate the `bind()` functions, just putting the body in the overloads of `SqlBase::<<`, and not using template for anything.

But then do we really need subclasses `Insert` and `Query`? Is there any functionality left asking for a subclass? The simplified example hides the info to make that decision. But if it's okay the example looks like this:

```
// ---- db.h -----
#include <string>
#include <vector>
#include <stdio.h>

namespace DB
{
class Row
{
public:
  std::string toString() const {return "Row";}
  // ... other method elided
};

const struct execI_t
{
} execInsert = {};
const struct execQ_t
{
} execQuery = {};

class Sql
{
public:
  Sql(const std::string & command) {}
  Sql & operator<<( int & value )
  {
    puts("bind int ");
    return *this;
  }
  Sql & operator<<( std::string & value )
  {
    puts("bind string ");
    return *this;
  }
  // ... other methods elided
  std::vector<Row> operator<<(const execQ_t &)
  {
    puts(" Query\n");
    return std::vector<Row>(1);
  }
  void operator<<( const execI_t &)
  {
    puts(" Insert\n");
  }
};

}

// ---- test.cpp -----
#include <fstream>
#include <iostream>
#include <string>
#include <vector>

#include "db.h"

int main()
{
  using namespace std;  // left for test only
//  using namespace DB; // could be used for
                        // the simple example
  using DB::execInsert;
  using DB::execQuery;
  using DB::Sql;
  using DB::Row;

  int id;
  string name;
  const string filename( "output.txt" );

  // populate id and name ...

  Sql( "insert into employee(name,id)"
    " values(%,%)" )
    << name  <<  id << execInsert;

  vector<Row> result = Sql( "select name "
    "from employee where id = %" )
    << id << execQuery;

  if(!result.empty())
  {
    string firstrow = result.at(0).toString();
    ofstream fstr( filename.c_str() );
    fstr << firstrow;
    fstr.close();
    if(!fstr)
      cerr << "output file failure\n";
  }
}
```

Certainly in the live version the implementation should do checks that all bindings happen, execution is last, the passed initialising string matches the type, and throw exceptions on execution failure. Also it must be aware of Sql injection attacks and avoid them. Note that the exec objects are made const and result is not accessed by `[0]` even in test. Use a check or member `.at()` avoiding undefined behavior in case 0 rows are returned. The functions are certainly not meant to be implemented in `db.h`, it's just to save space. The namespace `DB` is now fairly neutral, much less objections against using directive if fixed that way – or can be even considered for removing, if the mainstream usage would want non-prefixed forms anyway. The all-caps name could clash with a macro.

Let me finish with what I started: this was an excellent choice of example code.

Commentary

The code originally intrigued me because the paradigm of writing database code using the `<<` operator looked quite nice but the ambiguity problems were nasty. The two problems with the streaming operators are that :

1. Operators can be defined as free-standing functions or as member functions.

   With the existing C++03 standard this produces a subtle difference in behaviour with temporary stream objects.

   ```
   Stream() << object;
   ```

   This line compiles if and only if the relevant `operator<<` is a member function.

   However the next version of the standard contains 'r-value references', which *do* bind to temporary objects. In the current draft of the new standard streaming operators are declared to take the stream by r-value reference so the above code will now work in both cases. Unfortunately, there's still a debate about the binding of r-value references so it's not clear what the final standard will state.

2. Templates are a bit too universal for some things.

   The problem is that making the `operator<<` a template captures too many potential targets. However, if `operator<<` is not a template then the function loses type information – the return type can only be the type of the argument. (This matters because, for good reasons, the streaming operations for `exec` have different return types in the derived classes.)

   What is needed is a compromise – where the argument type is constrained but the return type can be the actual compile time type of the argument. This isn't directly supported by C++, although clever template tricks can sometimes get pretty close.

I'm left unsure after using code like this whether the pain is worth it – the streaming operator has an air of simplicity to it, but does it actually express intent more clearly than naïve methods calls?

The Winner of CC 55

Well there was only one entrant so the decision was easy …

I think Balog covered pretty well all the points with the code so I feel that he definitely deserves this issue's prize.

Code Critique 56

(Submissions to scc@accu.org by April 1st)

I'm writing a simple hash-map and I've got a couple of questions. Firstly, why does the line marked '1' in the header file need the word 'typename'? Secondly, my little test program doesn't quite work – the last line of output still shows the old value for 'key2'.

Please help the programmer answer their questions – and suggest some other questions they ought to think about too… the code is in Listing 2.

You can also get the current problem from the accu-general mail list (next entry is posted around the last issue's deadline) or from the ACCU website (http://www.accu.org/journals/).

This particularly helps overseas members who typically get the magazine much later than members in the UK and Europe.

Listing 2

```
// ---- hash.h -----
#include <algorithm>
#include <list>
#include <numeric>
#include <vector>

template <class Key, class Value>
class hashmap
{
public:
  hashmap( int size = 10 )
  : buckets(size) {}

  void set( Key key, Value value );
  Value get( Key key );

private:
  struct entry
  {
    Key key;
    Value value;
    bool operator==( Value key ) const
    {
      return key == key;
    }
  };

  int hashfun( Key s );
  std::vector< std::list<entry> > buckets;
  std::list<entry> bucket;
  typename std::list<entry>::iterator it; // 1
};

template <class Key, class Value>
void hashmap<Key, Value>::set( Key key,
  Value value )
{
  int hashval = hashfun( key );
  int thebucket = hashval % buckets.size();
  bucket = buckets[ thebucket ];
  if ( bucket.size() == 0 )
  {
    entry e;
    e.key = key;
    e.value = value;
    buckets[ thebucket ].push_back( e );
  }
```

Listing 2 (cont'd)

```
  else
  {
    it = std::find( bucket.begin(),
      bucket.end(), key );
    if ( it == bucket.end() )
    {
      entry e;
      e.key = key;
      e.value = value;
      bucket.push_back( e );
    }
    else
    {
      it->value = value;
   }
  }
}
template <class Key, class Value>
Value hashmap<Key, Value>::get( Key key )
{
  int hashval = hashfun( key );
  int thebucket = hashval % buckets.size();
  bucket = buckets[ thebucket ];
  it = std::find( bucket.begin(),
    bucket.end(), key );
  if ( it == bucket.end() )
  {
    return Value();
  }
  else
  {
    return it->value;
  }
}
template <class Key, class Value>
int hashmap<Key, Value>::hashfun( Key s )
{
  return std::accumulate( s.begin(),
    s.end(), 0 );
}

// ---- hash.cpp -----
#include <iostream>
#include <string>

#include "hash.h"

int main()
{
  hashmap<std::string, std::string> test;

  test.set("key1", "value1");
  test.set("key2", "value2");
  std::cout << "key1 = "
    << test.get( "key1" ) << std::endl;
  std::cout << "key2 = "
    << test.get( "key2" ) << std::endl;
  std::cout << "key3 = "
    << test.get( "key3" ) << std::endl;
  test.set("key2", "another_value2");
  std::cout << "key2 = "
    << test.get( "key2" ) << std::endl;
}
```

Desert Island Books

Paul Grenyer introduces Ian Bruntlett.

Desert Island books is deviating from its normal format this issue. Most people will already have realised that this series is not actually about books or music. It's about people.

Ian Bruntlett is well known to accu-general members, so when he sent me a piece describing his career to date, I was only too happy publish it as part of the series.

Ian Bruntlett

Ian Bruntlett is an IT volunteer worker at Contact, Morpeth, a mental health charity. Contact takes in unwanted computers, Ian refurbishes them and puts a lot of free software on. He's learning new skills, starting with basic stuff (Scribus, OpenOffice etc) and then moving on to more technical stuff. Contact allows its building to be used for the ACCU NE meetings.

Ian's first computer was a HP-33E programmable calculator. Then he moved to the Spectrum. Then he moved to the Sinclair QL and stayed there, writing articles and software. In particular his articles helped publicise the QPE – Qjump Pointer Environment. These days if you want a mouse you stick it in the back of your PC. In the QL days, you took the lid off your computer and removed a ULA (Uncommitted Logic Array), inserted a daughter-board and then reinserted the ULA.

After studying at Sunderland University for a BSc in I.T., he applied for Phd funding and failed to get it. So Ian moved into the world of commercial software development, spending the next 5 years working on LiBRiS Opac/Unity: a family of library book search engines and database building programs, written using Watcom C. At that time his most influential book was *Object Oriented Analysis* by Coad and Yourdon.

After that work dried up, Ian landed a job in Edinburgh, commuting between Berwick Upon Tweed and Edinburgh Haymarket. During that time he resumed hearing voices, in his case studying C++ and Design and getting feedback from voices in his head talking about programming. He was running code walk-throughs on the train with at least three voices in his head contributing.

Eventually Ian was made redundant, became very ill and ended up in a psychiatric hospital. Between 2001 and 2004, Ian became ill in a familiar routine – first you get a job, then you get stressed, then you go into hospital, then you lose job, and repeat. He looked on the internet for information about schizophrenics and their experiences. Finding little he started a blog: schizopanic.blogspot.com .

After going through this he was offered a post as a voluntary IT worker. Contact Morpeth (http://www.contactmorpeth.org.uk/) had been donated two PCs and a broadband connection by BT. So he ended up teaching people with mental health problems how to use computers. And after that had been going for a while, he reckoned that the people being trained preferred spending their money on cigarettes instead of IT equipment – so he sought permission to solicit donations of unwanted PCs. The scheme was approved and it has paid off handsomely. Ian created a 'software toolkit' (OpenOffice, Scribus etc, AVG anti virus, TuxTyper, TuxMath etc) of free software to put on PCs being donated to Contact's members. Free software has been very helpful in keeping costs down and he tends to put on programs that are available for Windows and Linux, with the eventual hope of moving some of Contact's general purpose PCs to either SuSE or Ubuntu Linux.

At the moment he is doing a bit of self-tuition – he's learning the basics of Ubuntu Linux, Windows XP and Vista and non-techie apps like

OpenOffice, GIMP, Scribus, Inkscape. After that he'll dip back into programming again.

The following books (amongst others) paved the way for Ian's career as a C++ programmer.

Multi-paradigm Design for C++ wasn't the first or last design book that he has read but has been the most influential – at the time he was reading it he was devouring every sentence, writing up study notes. So what was so important? Well, prior to that he just learnt language features and didn't have a framework to understand them. This book introduced him to 'Commonality Analysis' with its different dimensions (Structure, Name & Behaviour, Algorithm) and 'Variability Analysis' (with the same dimensions plus Binding Time and Defaults). Its coverage of 'Object Oriented Analysis' was the first time he read about OOA and actually had a 'Eureka!' moment. Domain Analysis (a well considered study resulting in a defined domain and dictionary) is the foundation for Domain Engineering (a software design discipline that focuses on the abstractions of a business (a domain) with the intent of reusing designs and artifacts). It is also influences Application Domain Analysis (where domain analysis is applied to the business/application domain) and Solution Domain Analysis (where the implementation technologies are examined). The book has 'Simple Mixing of Paradigms' and 'Weaving Paradigms Together' chapters, which heavily influenced him when he wrote a multi-paradigm library to map C++ objects on top of C. The book concludes with 'Augmenting the Solution Domain with Patterns', something he will get round to one day.

Design Patterns is a book he is still growing into. Every so often he notices a pattern in the book and thinks 'Ah! I've done that before' – the latest moment like that he had was with a C function called **`GetBookViaPosting`**, which was an object factory. He isn't working due to his illness and decided one day to help write free software. One benefit of such an activity is exposure, once more, to production C++ code and the spotting of the occasional patterns throughout the systems being worked on.

The *C++ Primer* gave him a good grounding in C++. Sure, there are more specific self-help C++ books out there (Scott Meyer's stuff, Herb Sutter's stuff) but this book gave him a good grounding and confidence in what was then modern C++. It is what he calls a 'Kilo-Page book': it's big, has exercises and is very thorough. Without this book, life writing C++ programs would have been a lot less pleasant. He found its coverage of classes and templates to be particularly helpful; however, he found that he needed a copy of Leen Ammeraal's *STL for C++ Programmers* to augment this book's coverage of algorithms.

Before he used the C++ Standard Library, he read Leen Ammeraal's *STL for C++ Programmers,* which got him going with C++'s standard library containers. Prior to using the STL, Ian rolled his own containers, typically arrays and linked lists – both of which can be handled with embarrassing ease with the STL. It's one thing to look at manual pages for the STL, its quite another to figure out how to use them without breaking something. His experience in writing his own containers helped him use and appreciate the STL.

Bookcase

The latest roundup of book reviews.

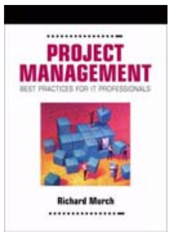
If you want to review a book, your first port of call should be the members section of the ACCU website, which contains a list of all of the books currently available. If there is something that you want to review, but can't find on there, just ask. It is possible that we can get hold of it.

After you've made your choice, email me and if the book checks out on my database, you can have it. I will instruct you from there. Remember though, if the book review is such a stinker as to be awarded the most un-glamourous 'not recommended' rating, you are entitled to another book completely free.

I must thank Blackwells and Computer Bookshop for their continued support in providing us with books.

Jez Higgins (jez@jezuk.co.uk)

Project Management: Best Practices for IT Professionals



By Richard Murch, published by Prentice Hall, ISBN: 978-0130219145

Reviewed by Paul Grenyer

Although I have actively, and sometimes passionately, resisted the move into any sort of management beyond team / technical leading for many years, I'm finding recently that I'm becoming more interested in project management. It's a sobering fact that where I am now I have a team and I need to manage them better.

Project Management by Richard Murch was (how shall I put it?) strongly recommended and presented to me by my current boss. It's a reasonably sized hardback book at 220 pages, plus appendices. The information on each page is, in most cases, both verbose and spread out, so it could have been a much smaller book. I would have scrubbed the final chapter on the internet altogether until I noticed the publication date of October 2000. It's a book of its time and therefore describes the more traditional project management techniques based around quite lot of documentation and rigid process. As such there is no mention of XP or Agile, neither of which could be ignored in a modern text. I read it cover to appendices in a little over a week and I learnt a lot.

There were a few things that irritated me about the book. Murch states very early on that all projects must cater for change. I couldn't agree with him more, but in several other places in the book he talks about eliminating scope creep. I may be being a little harsh as most of the scope creep Murch refers to is the 'wouldn't it be nice if' type. What he seems to have missed is that all scopes in software development creep. The reason they creep is that software engineers strive to give users what they asked for and, invariably, users ask for what they think they want, not what they actually need. Any well managed project will have the users involved from beginning to end (a point Murch does make) and as the users see what they're getting they're able to help direct the engineers to what they actually need and scope creeps. Scope creep could of course be eliminated by gathering

Bookshops

The following bookshops actively support ACCU (offering a post free service to UK members – if you ever have a problem with this, please let me know – I can only act on problems that you tell me about). We hope that you will give preference to them. If a bookshop in your area is willing to display ACCU publicity material or otherwise support ACCU, please let us know so they can be added to the list

- **Holborn Books Ltd** (020 7831 0022) www.holbornbooks.co.uk
- **Blackwell's Bookshop**, Oxford (01865 792792) blackwells.extra@blackwell.co.uk

Desert Island Books (continued)

Babylon 5 : A Call To Arms – core rules and fleet lists by Mongoose Publishing plus unofficial supplements by enthusiasts. (*Star Trek : TNG*, *Star Gate*). This was the first wargame that really took off for him (excepting *Greyhawk Wars*, which he played a couple of times). It isn't a perfect system and he's had to augment the books by creating a laminated: quick reference sheet for special actions and weapon traits. The game can be divided into: 1) core rules (they apply all the time) and 2) fleet lists (which at the moment are the B5 fleet lists published by Mongoose plus various fleet lists that have been devised by enthusiasts and uploaded onto the web with the tacit approval of Mongoose Publishing).

Traveller – Core Rulebook by Gareth Hanrahan, published by Mongoose Publishing. The possibilities of this game are mind blowing. While it is set during the days of a galactic empire, the 'Third Imperium of Mankind', it can be used to play in almost any science fiction setting. He'd one day like to run scenarios in, say, the *Martian Trilogy* by Kim Stanley Robinson, the merchant setting of C. J. Cheryyh or the *Darkover* setting by Marion Zimmer Bradley. This is an old game given new life. Mongoose have declared on their blog that they are going to support *Traveller* for the long term.

The Kick Inside by Kate Bush is the first record Ian bought with his own money. And yes, it was a real record – made of vinyl and spinning around at 33 1/3 rpm. He initially bought it for the *Wuthering Heights* single. It was second hand and he paid £1.50 for it. It had loads of hisses and scratches, so much so that when he bought the CD version, he thought the missing hisses were a defect. ■

What's it all about?

Desert Island Disks is one of BBC Radio 4's most popular and enduring programmes:

http://www.bbc.co.uk/radio4/factual/desertislanddiscs.shtml

The format is simple: each week a guest is invited to choose the eight records they would take with them to a desert island.

I've been thinking for a while that it would be entertaining to get ACCU members to choose their Desert Island Books. The format will be slightly different from the Radio 4 show. Members will choose about 5 books, one of which must be a novel, and up to two albums. The programming books must have made a big impact on their programming life or be ones that they would take to a desert island. The inclusion of a novel and a couple of albums will also help us to learn a little more about the person. The ACCU has some amazing personalities and I'm sure we only scratch the surface most of the time.

Each issue of CVu will have someone different. If you would like to share your Desert Island Books please email me: paul.grenyer@gmail.com.

Next issue: Ewan Milne picks his desert island books.

requirements and then isolating the users until the project is complete, but then the users won't have what they need.

The book also shows its age when describing testing. Unit Testing no longer refers to testing a unit of work. It is an automated test testing a single unit such as a class. These test should be written by the software engineer and not left to another party following development.

I haven't encountered Rapid Application Development (RAD) in the project management sense before. From Murch's description it sounds like the only teams it would suit are those that are simply stringing together existing components, where there is no learning to be done and very little to zero chance of scope creep. This seams totally unrealistic to me. Software engineers innovate and to innovate they have to learn. When people are learning they make mistakes and RAD doesn't allow for mistakes.

That said, let me reiterate that I learnt a lot and there is plenty of good advice in this book. As I read it I made a list of the things to remember and to try and make use of. I ended up with 19 points and page numbers. To my surprise a lot of these did revolve around documentation, but I still feel it should be minimal and I'm still undecided about how much should be hard-copy and how much electronic.

Murch states that project managers should stay positive (P.15) about all aspects of the project while, at the same time, be able to handle often relentless stress (P.18). This is a very accurate observation that all project managers should bear in mind.

A lot of, but not all, software engineers and managers are aware of the risks that might affect their projects. Murch talks about risk management objectives (P.163) and documenting and constantly reviewing risks that effect the project. The scoring system and examples presented are sensible and I can see them being useful. Once identified, all risks should have a risk management plan (P.166). This all makes perfect sense, especially when you consider 'the first step in avoiding a trap is knowing of its existence'. This way there are less surprises for all concerned.

Software engineers are often left to battle on with a problem on their own, be it pride or the illusion that someone else's time is better spent on a separate problem or on development. Murch points out that problems are solved quicker and more easily when more than one person investigates the problem (divide and conquer P.176). He also describes a repeatable problem solving methodology (P.177) that helps the team to understand and document the problem for future reference, plan and implement a solution.

Problems do occur in projects. In most cases these problems are fixed and the team assigns them to history and moves on. This kills the learning process as soon as the problem is fixed, which leaves it liable for repetition in the future. Murch describes 'Lessons Learned' reviews (P.28) which are a type of post mortem designed to help the team learn from problems and mistakes to help avoid them in the future.

All projects, even Agile and XP projects need a project plan (P.41) and no project should be started without one. Projects should be divided up into phases and/or milestones with a phase check list (P.64) so that it is clear to the team and to project and senior management if a phase has been completed successfully. Regular project review reports (P.32) enable the team, the project manager and senior managers to monitor and understand the progress of the project. Projects also need a set of standards (P.29) that are reviewed regularly to make sure that everyone is producing the right level of quality.

Murch explains, as we all know, that people are the most important part of any project. It is important to retain staff and make them feel secure. Murch suggests that the best way of doing this is with incentive packages, usually consisting of a high salary, options, dental plans etc. Although I agree (what software engineer wouldn't) the people you work with are often the biggest factor. Therefore, when expanding the team, it is vitally important to find people who will fit into the team, not just those that are the right price or have the right level of technical skill. Evaluating the skills within the team over time (P.24) is important. It helps make sure that team members learn and maintain the skills that are going to be of most use to the team currently and in the future.

Murch describes two roles that would be useful in any team: a scribe (P.157) who is responsible for recording and collating discussion in any team meeting (formal or otherwise) and a release manager (P.193) who is responsible for overseeing releases, release planning and managing any problems that arise.

I wouldn't recommend this book as an absolute or only guide to project management, especially as it is so out of date, but it is a good place to start if you are going on to read other things. As for me, I have *Agile Project Management: Creating Innovative Products* (ISBN: 978-0321219770) lined up next.

Programming Microsoft Visual C# 2008: The Language

By Donis Marshall, published by Microsoft Press, 746 pages, ISBN: 0-7356-2540-9

Reviewed by James Talbut

Not recommended, but not without merit.

This is an update to the 2005 edition, also by Donis Marshall, that was substantially shorter.

The book attempts to be a reference, presented in prose as a tour through all the features of the C# language, covering only those aspects of .NET that are necessary for this tour. This results in some coverage that is idiosyncratic, for example a very brief coverage of using LINQ (a language feature) to access databases, without any coverage of any other techniques for database access. Despite this, the coverage presented by the book is generally reasonably thorough, it's certainly enough to provide a thorough introduction to C#. That's the good news.

The bad news is that the book cannot decide who it is trying to present this information to. Terms such as 'the stack' and 'the heap' are used in the first chapter without any explanation, but then in Chapter 3 there is an inane explanation of inheritance using people as an example. There are also pointless statements which seem intended purely to increase the word count: 'This property returns the rank of the array, which is the number of dimensions. For example, a two-dimensional array has a rank of two.' I would hope that anyone capable of coping with this book could work that out for themselves – there are similarly redundant examples elsewhere that occupy entire paragraphs, but they are not suitable for quoting.

Donis also presents some questionable statements as facts, without consideration of counter examples. Chapter 1, with reference to short circuit evaluation of Boolean expressions, contains the statement 'Without disciplined coding practices, short-circuiting might cause unexpected side effects'. To my mind, any potential issues here are caused by the use of expressions with side effects in Boolean expressions, not by the use of short-circuiting, but no explanation of this is provided at all.

My final complaint is that there is a distinct dislike of C++ presented in the book, from Chapter 7: 'The CLR performs an intelligent expansion of generic types, unlike C++, which blindly expands parameterized types', which is, at best, presented as propaganda, with no consideration of the merits of compile time evaluation in C++. There are a number of examples of statements such as this, which will simply annoy any existing C++ developers.

Despite the wealth of information in this book, it has taken me some three months to wade through to the halfway point. At this point I have decided that I have more interesting books to read, though I will still occasionally look up topics in here for reading in the smallest room.

Object Oriented Construction Handbook: Developing Application-Oriented Software with the Tools and Materials Approach

By Heinz Zullighoven (with many contributing authors), published by Morgan Kaufman, ISBN: 1558606874

Reviewed by John Penney

Over the course of several large-scale German and Swiss enterprise-wide application developments, Zullighoven and co. evolved a development methodology for enterprise applications that they have termed the 'Tools and Materials' approach – or T&M. This book was first published in Germany in 1998, this is the first English edition in 2005.

The T&M approach characterises the artifacts the manipulated by the end-user as either 'tools' (such as a ruler) or 'materials' (such as a spreadsheet). Materials offer interfaces that tools can use, and this metaphor extends from the UI design through construction of the domain model to class-level design. The key features of T&M seem to be an extensive use of real-world metaphors, and a great emphasis on mapping the application development to the business and problem domain. In this regard, it seems there's a great deal of overlap between the T&M approach and Eric Evans' much more well-known Domain Driven Design.

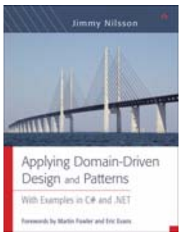
The fundemental flaw here – and the reason why this book will gather dust – is that this is very academic, dense and wordy – a weighty 600 pages. Understanding the complex sentences is made much more difficult by the questionable English. I found whole paragraphs that were grammatically correct but which I struggled to make sense of. Doomed attempts to add clarity through examples only resulted in unintentional comedy: 'if someone says a sentence like "that was close to the limit," who would think of a borderline between two countries?' Err... no-one, Heinz? 'A pen is a tool [when being used to write] but not when [used with] a sharpener to sharp it'. Obviously the quill is still widely used in German software engineering circles.

There is undoubtedly a good deal of wisdom buried in this book: the formal definitions of patterns, frameworks and components and the catalogues of design metaphors and modelling artifacts are impressively thorough. However, for this to be at all useful to a lesser mortal lacking superhuman powers of concentration, an 'Introduction to T&M' (written by someone else!) would be essential. Lacking this, I'll stick with Eric.

Applying Domain Driven Design and Patterns With Examples in C# and .Net

By Jimmy Nilsson, published by Addison Wesley, ISBN: 0-321-26820-2

Reviewed by Omar Bashir



Recommendation: Verbose and anecdotal but contains relevant knowledge and interesting case studies.

Going through this book was a mission. First seven pages of the book contain praise for the book with the last one from author's eight year old son, 'Dad, do you really think someone will read it'. As I went deeper into the book, that remark frequently kept coming back to mind.

I will give credit to author's sincere intentions in adopting an informal approach to explaining the design and test driven development of software based on domain models. However, his explanation of the design and implementation process becomes increasingly anecdotal leading to substantial and unnecessary verbosity. The information contained in this book could have easily been contained in a book less than half its size. The text contains many unnecessary, unrelated and uninteresting comments and notes that break its flow making it considerably challenging to focus on the subject.

The author consistently advocates moderate application of a number of aspects of agile software development rather than a zealous (over) application of methodologies. He stresses an informed and an experience based adaptation of various methodologies. Although he does not advocate an upfront formal design but he does encourage a limited and informal upfront design of an infrastructure agnostic domain model that is progressively refined through TDD. The author also suggests reusing successful architectural styles and frameworks within a development environment. In fact he strongly recommends developing and using a generic framework that can be customized on project/product basis as these provide consistency in development, reduce per project learning curve and also reduce overall time to market.

The book contains a number of sections by invited authors provide interesting introduction to TDD using mocks and stubs, design approaches to adopt and approaches to developing UI to support domain driven applications. An appendix discusses variations in domain models considering the requirements of development exercise being followed in the book as viewed by three invited authors. Their relatively moderately formal and focused writing style makes for refreshing breaks within the overall text.

The book refers considerably to patterns and techniques authored by Eric Evans in his *Domain Driven Design* book and by Martin Fowler in his *Patterns of Enterprise Application Architecture* book. A prior study of these books or the referenced topics helps understanding this book. Furthermore, this book also expects familiarity with GoF patterns and TDD. The latter, however, is described and illustrated in detail through various examples.

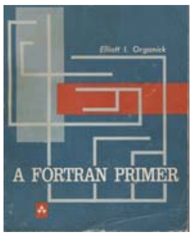
The book references mostly open source .Net technologies (e.g. Spring.Net) without mentioning any application of the described approaches using Microsoft technologies like the Enterprise Services, .Net web services etc. Author's use of C# is unimpressive with no use of and only a passing reference to generics and a limited discussion on the application of attributes (annotations in Java) in domain related or support code. Use or discussion of the .Net framework is also non-existent. The author also expresses strong opinions against XML suggesting that suitable tools did not exist for XML editing and that most IDEs cannot associate XML to the source code and therefore cannot assist in updating XML while refactoring the code. While the latter may be true, the former (existence of comprehensive XML tools) may not entirely be true. Most new IDEs (Visual Studio, Eclipse etc.) contain reasonable XML editors while a considerable number of commercial and open source XML tools are also available.

Finally, from the infrastructure perspective, the book only discusses persistence. It would have been interesting if the author could have generalized these concepts for application to other infrastructure related aspects such as messaging and communication.

A FORTRAN Primer

By Elliott I. Organick, published by Addison-Wesley, 183 pages, 'Copyright (C) 1963' but 'Philippines Copyright 1965', 'Fourth printing, March 1966'

Reviewed by: Colin Paul Gloster



I have become involved with a number of Fortran users, so I read this book, as it is for beginners and was short enough to be read quickly; even though it is dated 1965 and deals with out-of-date Fortran II dialects (with a chapter on Fortran IV). This might lead one to suspect that it is of little use, and admittedly, some of the Fortran II language documented herein has been removed from newer versions of Fortran.

Nonetheless, it was worthwhile to confirm for myself the miseducation directed towards Fortran users, the aftermath of which persists circa four decades later. For example, a single 318-line Fortran file in a 2003 code available online (Fortran users treat the word code as a countable noun), has 45 `GO TO` statements, jumping (if they are ever reached), to 26 distinct numerical statement labels (which are called line numbers in BASIC). For the benefit of readers of this review who are new to programming, I point out that it is typically a bad idea to use `GO TO`.

The flow of English words in the `GO TO` chapter fittingly hops back and forth between the columns in a confusing manner dissimilar to the preceding chapters. I eventually noticed that, unlike dialects of BASIC which I had used, numerical statement labels do not need to be in sequential order in Fortran. One of the many examples of this feature in the book is Statement 53 followed by Statement 10.

Hard-coded numbers (magic numbers), instead of symbolic names are another bad practice found in this book; the first program relies on an undefined symbolic constant. Another item of note is the first figure in Chapter 3 – referenced as Fig. 3-1 in the text but Fig. 3-3 in the caption.

The first mention of user-defined functions appears in a single paragraph in Chapter 6, before their next appearance in Chapter 11. In decent programming languages, user-defined functions are important and should be ingrained into learners' habits from an early stage. It is easy

ACCU Information

Membership news and committee reports

View From The Chair

Jez Higgins
chair@accu.org

As I type, the Chair's chair is seat D48 of 390-013 Virgin Spirit[1], being the 17:23 from London Euston. It's overwarm and rather cramped. The view, what I can make out in the little light that remains, is relative featureless arable farmland that occupies the gap between London and Milton Keynes. Fortunately it's whipping by at 100 miles an hour and, if I had daylight to see it, I know it would get significantly prettier as we head northwards. I usually spend my working hours hanging around in my attic, but I ventured out onsite today because I've been appointed 'publishing guru'. At work, we're installing and commissioning a new content management system which will, according to the consultant we've engaged, work splendidly once we get it working. Getting it working is the tricky part. Accordingly, several teams have been reorganised around specific areas of functionality, development practices have been given a bit of a kick up the backside, and so on. Publishing, that is getting the content and code out development and editorial and off onto a website where customers can get what they're paying for, cuts right across the neat team boundaries, and so I have something of a roving brief. It promises, by turns, to be both frustrating and rather interesting. Being handed this position might simply be the result of not being in the room when the jobs were handed out, but it was sold to me in a much more positive way. 'You know', I was assured, 'lots of stuff about a lot of things.' I countered that while my knowledge might be broad, it was relatively shallow. Apparently, breadth was the overriding factor. Being a gadfly generalist is, it seems, something of speciality these days.

In a roundabout way ACCU has helped feed my position as generalist-in-chief. I've written often about this before, but ACCU's journals, mailing lists, and conferences, continually throw up new things to think about. While the import of something is rarely apparent at the time, merely having some little snippet or clue or idea lurking somewhere deep in your memory is enough to get you started. We all need a bit of a leg up now and again, and ACCU is capable of giving a pretty hefty hoik. Everybody should have that to call upon, so I hope you'll take up David's recruit-a-member challenge. New members will help us, and we'll help them. Some of my over-eager work colleagues might call that a 'win-win', but I prefer to think of it in less mercenary terms as 'a pleasant thing to do for a friend'.

ACCU's AGM will shortly be upon us, taking place on the Saturday 25 April during the conference at the Barcelo Hotel in Oxford. (You don't have to be attending the conference to attend the AGM.). One important item on the agenda each year is the election and reelection of the committee and the officers. You can get involved in the running of the organisation at anytime, simply by offering, but the AGM is an obvious start point. If you have an idea for something ACCU could or should be doing or for something it's doing now that could be done better, then consider standing for the committee. Being on the committee needn't represent a huge time investment, but as a volunteer-run organisation everybody's contribution makes a difference. Do drop me a line if you'd like to discuss an idea you have, or to find out more.

[1] And I know this because ... http://is.gd/j9Ci

Membership

Mick Brooks
accumembership@accu.org

First an apology to those of you who emailed accumembership@accu.org since December and didn't get a prompt reply. Google's mail filter took exception to some (but not all) ACCU-related messages and marked them as spam. By the time I noticed (it was an even-more-than-usually disjointed thread on the accu-general list that gave it away) at the end of January there was a significant backlog of messages that I needed to respond to. I'm now caught up, so if you wrote to me and still haven't had a response could you drop me another line?

ACCU currently has just under 810 members – the same number that we had at the end of August. I expect this to increase in the next month as people join to book for the annual conference, but, as David Carter-Hitchin stressed recently, we do need your help to ensure we get new members. I'm sure you all have colleagues and friends who would be interested in membership of ACCU – you just have to let them know we're here. If you could make use of a few extra magazines or flyers for your coffee room let me know: I always have spares that I can send out.

As always, please send any questions or suggestions about membership and renewals to me at accumembership@accu.org. (I'll be checking my spam folder from now on, I promise.)

Publicity

David Carter-Hitchin
publicity@accu.org

Help the ACCU!

ACCU membership numbers have reduced quite significantly lately, and my guess is that this is related to the economic downturn. In order to counteract this we need to boost our numbers by telling everyone about the ACCU. One challenge this year is to recruit one member from your friends and work colleagues. If everyone did that the ACCU would double in size. The big project for me this year is going to be to get an ACCU leaflet into every library in the UK and in other countries where the ACCU have a presence. This sounds like a time consuming task but actually it's very simple, but I do need your help. Most libraries have a distribution function for leaflets and other information, so you can give a stock of leaflets to one library and they will distribute them around the local county or region. All I need you to do at this stage is to mail me your location. I will then pick one person from each county/region and ask them to visit their local library to find out how many leaflets they would need. I will then send you the leaflets which you take to your library. Easy eh? So don't delay – please sign up for this by e-mailing me at publicity@accu.org.

Adverising

Seb Rose
ads@accu.org

A couple of our advertisers are reaching the end of their campaigns over the next month or two, so there may be some spaces available for new web campaigns. We run a maximum of five web campaigns at a time (to guarantee our advertisers decent exposure), and we're generally fully booked. If you know of any organisations that might benefit from advertising in our journals and on the website, please forward their contact details to ads@accu.org

The 21st AGM

Notice is hereby given that the 21st Annual General Meeting of The C Users' Group (UK) publicly known as ACCU will be held during the lunchtime break on Saturday 25th April 2009 at the Oxford Barceló Hotel, (formerly the Oxford Paramount Hotel), Godstow Road, Oxford, OX2 8AL, United Kingdom.

Current agenda

1. Apologies for absence
2. Minutes of the 20th Annual General Meeting
3. Annual reports of the officers
4. Accounts for the year ending 31st December 2008
5. Election of Auditor
6. Election of Officers and Committee
7. Other motions for which notice has been given.
8. Any other Annual General Meeting Business (To be notified to the Chair prior to the commencement of the Meeting).

The attention of attendees under a Corporate Membership is drawn to Rule 7.8 of the Constitution:

> ... Voting by Corporate bodies is limited to a maximum of four individuals from that body. The identities of Corporate voting and non-voting individuals must be made known to the Chair before commencing the business of the Meeting. All individuals present under a Corporate Membership have speaking rights.

Also, all members should note rules 7.5:

> Notices of Motion, duly proposed and seconded, must be lodged with the Secretary at least 14 days prior to the General Meeting.

and 7.6:

> Nominations for Officers and Committee members, duly proposed, seconded and accepted, shall be lodged with the Secretary at least 14 days prior to the General Meeting.

and 7.7:

> In addition to written nominations for a position, nominations may be taken from the floor at the General Meeting. In the event of there being more nominations than there are positions to fill, candidates shall be elected by simple majority of those Members present and voting. The presiding Member shall have a casting vote.

For historical and logistical reasons, the date and venue is that of the last day of the ACCU Spring Conference. Please note that you do not need to be attending the conference to attend the AGM.

(For more information about the conference, please see the web page at <url:http://accu.org/conference>.)

More details, including any more motions, will be announced later. A full list of motions and electoral candidates will be supplied at the meeting itself.

Alan Bellingham
Secretary, ACCU

Bookcase (continued)

to condemn the author for presenting the topic so late, but unfortunately Fortran is not a decent programming language, partially because Fortran compilers are not required to detect actual parameters that do not correspond to the formal parameters. (It is possible to overcome this deficiency in Fortran 90 onwards ... if one knows how to).

It has been claimed on a number of web pages that the **`COMPLEX`** and **`DOUBLE PRECISION`** modes (types) were added in Fortran II, but in this book they are mentioned only in the Fortran IV chapter. Perhaps these dialectal names are unofficial and have ill-defined boundaries, or perhaps someone was mistaken.

The author clearly went to a lot of effort to make this book comprehensible to beginners to programming. Such readers who must use Fortran (for example, those who are forced to in a course which is predominantly unrelated to computers), should not use this book. It may be more worthwhile for someone used to programming who wants a quick overview of Fortran without expecting everything to be as it is in a recent Fortran version.