***Thợ lành nghề #8: Kiểm thử là một dạng tài liệu (Dịch vụ Socket 3)***

*Ở bài này tay học việc của chúng ta học được một điều: các kiểm thử có mục đích phục vụ lớn hơn là chỉ đơn thuần để chứng minh là mã nguồn chạy được: Kiểm thử là một dạng tài liệu thực hành và giáo dục.*

Bạn có thể tải mã nguồn mà chúng ta đã có ở phần [*Thợ lành nghề #7: Một lần không đủ (Dịch vụ Socket 2)*](http://tapchilaptrinh.wordpress.com/2013/05/03/tho-lanh-nghe-6-mot-lan-khong-du-socket-service-1/) *[ở đây.](http://www.objectmentor.com/resources/craftman/SDSocketServerR2_ManyConnections.zip)*

*Ngày 14 tháng 11 năm 2002.*

Thang máy tầng 17 lại hỏng nên tôi phải dùng trụ tuột. Trong lúc tụt xuống, tôi bắt đầu ngẫm nghĩ đến chuyện đáng ghi nhận là dùng kiểm thử như một thứ đồ nghề thiết kế. Chìm đắm trong suy nghĩ, tôi hơi vô ý nên va cùi chỏ vào trụ thang với sức dội Coriolis [\*]. Khi tôi gặp Jerry trong phòng thí nghiệm nó vẫn còn đau nhói.

“Mày sẵn sàng thử cái kiểm thử dùng để gửi thông điệp ‘Hello’ qua socket chưa?” Gã hỏi.

“Chắc chắn rồi”, tôi đáp.

Chúng tôi bỏ phần chú thích của phương thức testSendMessage.

public void testSendMessage() throws Exception {

SocketService ss = new SocketService();

ss.serve(999, new HelloServer());

Socket s = new Socket(&quot;localhost&quot;, 999);

InputStream is = s.getInputStream();

InputStreamReader isr = new InputStreamReader(is);

BufferedReader br = new BufferedReader(isr);

String answer = br.readLine();

s.close();

assertEquals(&quot;Hello&quot;, answer);

}

“Như dự đoán, đoạn này không biên dịch được” Jerry phát biểu. “Mình cần phải viết cái HelloServer.”

“Tôi nghĩ mình biết phải làm gì,” tôi trả lời. “HelloServer là lớp con của SocketServer và dùng phương thức serve() để gởi thông điệp ‘Hello’ qua socket.”

Tôi vớ lấy bàn phím và điều chỉnh mã theo kiểm thử trong TestSocketServer.java như sau:

class HelloServer implements SocketServer {

public void serve(Socket s) {

try {

OutputStream os = s.getOutputStream();

PrintStream ps = new PrintStream(os);

ps.println(&quot;Hello&quot;);

} catch (IOException e) {

}

}

}

Ðoạn này biên dịch được và các kiểm thử đều đạt ngay lần đầu.

“Ngon lành,” Jerry nói. “Bây giờ chúng ta có thể gửi một thông điệp qua socket.”

Tôi biết Jerry bắt đầu nghĩ đến chuyện tái cấu trúc và tôi muốn qua mặt gã. Xem xét đoạn mã kỹ lưỡng, tôi nhớ gã có đề cập đến vấn đề trùng lặp.

“Có một số mã trùng lặp trong các kiểm thử đơn vị,” tôi nói. “Trong mỗi kiểm thử mình tạo và đóng SocketService. Chúng ta nên bỏ nó đi.”

“Tinh mắt lắm!” Jerry phán. “Hãy dời nó vào các hàm setUp và tearDown.” Gã tóm lấy bàn phím và thay đổi như sau:

private SocketService ss;

public void setUp() throws Exception {

ss = new SocketService();

}

public void tearDown() throws Exception {

ss.close();

}

Sau đó gã bỏ toàn bộ các dòng lệnh **ss = newSocketService();** và **ss.close();** trong ba kiểm thử.

“Xem được hơn đó,” tôi nói. “Hãy thử xem mình có thể gửi một thông điệp ngược lại không.”

“Tao cũng nghĩ y như vậy,” Jerry trả lời. “Và tao có một cách làm chuyện đó.” Gã bắt đầu viết một trường hợp kiểm thử mới:

public void testReceiveMessage() throws Exception {

ss.serve(999, new EchoService());

Socket s = new Socket(&quot;localhost&quot;, 999);

InputStream is = s.getInputStream();

InputStreamReader isr = new InputStreamReader(is);

BufferedReader br = new BufferedReader(isr);

OutputStream os = s.getOutputStream();

PrintStream ps = new PrintStream(os);

ps.println(&quot;MyMessage&quot;);

String answer = br.readLine();

s.close();

assertEquals(&quot;MyMessage&quot;, answer);

}

“Eo ôi! Tởm thế,” tôi cằn nhằn.

“Ừ, đúng thật,” Jerry thú nhận. “Hãy làm cho nó chạy cái đã rồi mình dọn dẹp nó sau. Chúng ta không muốn mớ lộn xộn đó ở đây lâu! Mày biết tao định làm gì phải không?”

“Vâng,” Tôi trả lời. “EchoService sẽ nhận một thông điệp từ socket và gửi ngược lại ngay. Bởi thế, đoạn kiểm thử của ông chỉ gửi MyMessage; rồi đọc nó lại.”

“Ðúng rồi. muốn thử ngó ngoáy code cho phần EchoService không?”

“Tất nhiên,” tôi nói một cách hăm hở.

class EchoService implements SocketServer {

public void serve(Socket s) {

try {

InputStream is = s.getInputStream();

InputStreamReader isr = new InputStreamReader(is);

BufferedReader br = new BufferedReader(isr);

OutputStream os = s.getOutputStream();

PrintStream ps = new PrintStream(os);

String token = br.readLine();

ps.println(token);

} catch (IOException e) {

}

}

}

“Oái,” tôi nói. “Lại thêm một mớ mã xấu xí. Mình cứ tạo các đối tượng của PrintStream và BufferedReader từ socket. Chúng ta cần phải dọn dẹp mới được.”

“Mình sẽ làm chuyện đó ngay sau khi mấy cái kiểm thử chạy ngon lành,” Jerry đáp, trong khi nhìn tôi có vẻ kỳ vọng.

“Oh!” tôi rú lên.

“Tôi quên chạy cái kiểm thử.” Xấu hổ, tôi nhấn nút kiểm thử và theo dõi nó chạy.

“Cũng không khó lắm,” tôi nói. “Bây giờ hãy vứt phần mã xấu xí ấy đi.” Tôi bổ sung thêm một số hàm cho EchoService.

class EchoService implements SocketServer {

public void serve(Socket s) {

try {

BufferedReader br = getBufferedReader(s);

PrintStream ps = getPrintStream(s);

String token = br.readLine();

ps.println(token);

} catch (IOException e) {

}

}

private PrintStream getPrintStream(Socket s) throws IOException {

OutputStream os = s.getOutputStream();

PrintStream ps = new PrintStream(os);

return ps;

}

private BufferedReader getBufferedReader(Socket s) throws IOException {

InputStream is = s.getInputStream();

InputStreamReader isr = new InputStreamReader(is);

BufferedReader br = new BufferedReader(isr);

return br;

}

}

“Ðoạn này cải tiến phương thức EchoService,” Jerry nói, “nhưng nó làm rối lớp lên một ít. Hơn nữa, nó không giúp gì cho hàm testRecieveMessage, đó cũng là một điểm không đẹp. Thử nghĩ getBufferedReader và getPrintStream có nằm đúng chỗ không?”

“Ðây sẽ là vấn đề lặp lại,” tôi nói. “Ai muốn dùng SocketService phải sẽ chuyển socket thành BufferedReader và PrintStream.”

“Chính là câu trả lời!” Jerry đáp lại. “Các phương thức getBufferedReader và getPrintStream quả thực thuộc về SocketService.”

Tôi chuyển hai hàm vào lớp SocketService và thay đổi EchoService theo cách đó.

public class SocketService {

//[...]

public static PrintStream getPrintStream(Socket s) throws IOException {

OutputStream os = s.getOutputStream();

PrintStream ps = new PrintStream(os);

return ps;

}

public static BufferedReader getBufferedReader(Socket s) throws IOException {

InputStream is = s.getInputStream();

InputStreamReader isr = new InputStreamReader(is);

BufferedReader br = new BufferedReader(isr);

return br;

}

}

class EchoService implements SocketServer {

public void serve(Socket s) {

try {

BufferedReader br = SocketService.getBufferedReader(s);

PrintStream ps = SocketService.getPrintStream(s);

String token = br.readLine();

ps.println(token);

} catch (IOException e) {

}

}

}

Các kiểm thử đều chạy. Giữ vững tình hình, tôi nói: “bây giờ tôi hẳn có thể sửa đổi phương thức testReceiveMessage luôn.” Trong lúc Jerry quan sát, tôi thay đổi phần này như sau:

public void testReceiveMessage() throws Exception {

ss.serve(999, new EchoService());

Socket s = new Socket(&quot;localhost&quot;, 999);

BufferedReader br = SocketService.getBufferedReader(s);

PrintStream ps = SocketService.getPrintStream(s);

ps.println(&quot;MyMessage&quot;);

String answer = br.readLine();

s.close();

assertEquals(&quot;MyMessage&quot;, answer);

}

“Ừa, coi được hơn đó,” Jerry nói.

“Không chỉ như vậy mà các kiểm thử đều đạt,” tôi nói. Thế rồi tôi nhận thấy thêm một điều. “Ui, có thêm một cái nữa trong testSendMessage.” Tôi sửa cái đó luôn.

public void testSendMessage() throws Exception {

ss.serve(999, new HelloServer());

Socket s = new Socket(&quot;localhost&quot;, 999);

BufferedReader br = SocketService.getBufferedReader(s);

String answer = br.readLine();

assertEquals(&quot;Hello&quot;, answer);

}

Các kiểm thử vẫn chạy. Tôi ngồi tẩn mẩn xét lại lớp TestSocketServer, tìm xem có gì loại bỏ được không.

“Mày xong chưa?” Jerry hỏi.

Tôi gật đầu.

“Tốt,” gã đáp lại. “Tao đã bắt đầu thấy khói bay ra khỏi tai mày đó. Tao nghĩ đã đến lúc cần nhanh chóng nghỉ ngơi.”

“OK, nhưng trước hết tôi có một câu hỏi.”

“Nói đi.”

Tôi nhìn qua nhìn lại Jerry và mã nguồn vài giây và nói “Chúng ta chẳng thay đổi tí nào cái SocketService. Mình thêm testSendMessage và testRecieveMessage và cả hai đều chạy. Mình lại tốn rất nhiều thời gian để viết mấy cái kiểm thử và lo chuyện tái cấu trúc. Làm như thế có lợi gì cho mình nhỉ? Chúng ta chẳng thay đổi mã nguồn chính của sản phẩm gì hết!”

Jerry nheo mày và ném cho tôi một vấn đề không thể hiểu nổi. “Cái mày vừa nói khá thú vị. Mày có nghĩ chúng ta đã lãng phí thời gian?”

“Điều đó không giống sự lãng phí, bởi vì chúng ta đã chứng tỏ với bản thân mình rằng mày có thể gửi và nhận các thông điệp. Tuy nhiên, chúng ta đã viết và tái cấu trúc rất nhiều code mà chẳng giúp được gì thêm cho mã nguồn chính của sản phẩm cả.”

“Bộ mày không nghĩ là getBufferedReader và getPrintStream đáng được đưa vào sản phẩm?”

“Mấy cái này khá tầm thường; chúng chỉ hỗ trợ cho mấy cái kiểm thử mà thôi.”

Jerry thở dài và nhìn đi hướng khác một phút. Sau đó hắn nhìn lại tôi và nói “Nếu mày dính vào dự án này, tao chỉ cho mày mấy cái kiểm thử, xem chúng dạy mày được những gì?”

Đó là một câu hỏi khác lạ. Phản ứng đầu tiên của tôi là trả lời bằng cách nói rằng chúng sẽ chỉ cho tôi thấy rằng tác giả đã muốn chứng minh mã nguồn của họ chạy tốt. Nhưng tôi kìm giữ ý nghĩ đó lại. Jerry sẽ không hỏi tôi câu hỏi đó nếu anh ta không muốn tôi suy nghĩ cẩn thận về câu trả lời.

Tôi học được gì từ mấy cái kiểm thử đó? Tôi học được cách tạo SocketService và gắn kết SocketServer tới đó. Tôi cũng học được cách gửi và nhận thông điệp. Tôi học được tên và vị trí của các lớp trong khung làm việc và cách sử dụng chúng.

Vì vậy tôi tập trung hết can đảm và nói: “Ý ông mình viết mấy cái kiểm thử này để làm ví dụ cho những người khác?”

“Ðó là một phần lý do, Alphonse. Những người khác sẽ có thể đọc mấy cái kiểm thử này và xem cách làm việc của mã nguồn. Họ cũng có thể làm việc qua nhưng lý giải của chúng ta. Hơn thế, họ sẽ có thể biên dịch và thao tác các kiểm thử này và chứng minh với chính họ rằng cách lý giải của chúng ta đáng thuyết phục. Còn nhiều điều hơn thế nữa,” gã nói tiếp, ” nhưng chúng ta để dành vấn đề này cho một dịp khác.”

Cùi chỏ tôi vẫn còn đau nhức nên tôi mừng là thang máy đã được sửa. Trong khi đi thang máy, tôi không ngừng nghĩ ngợi: “Kiểm thử là một dạng tài liệu-có thể biên dịch được, có thể thao tác và luôn luôn đồng bộ.”

*[\*] Còn có tên gọi là Coriolis effect gọi theo tên của kỹ sư – nhà toán học Pháp Gustave-Gaspard Coriolis. Ở đây dường như tác giả mô tả hành động tay học việc ôm cột tụt xuống và trong khi tụt xuống, anh ta ở trạng thái xoay vòng trên cột nên bị “Coriolis force”. Xem thêm chi tiết ở:* [*http://zebu.uoregon.edu/~js/glossary…fect.html*](http://zebu.uoregon.edu/~js/glossary...fect.html)*. và* [*http://satftp.soest.hawaii.edu/ocn620/coriolis/*](http://satftp.soest.hawaii.edu/ocn620/coriolis/) *– chú thích của người dịch.*

**Bài tiếp:** [Thợ lành nghề #9: Những thread nguy hiểm (Dịch vụ Socket 4)](https://codegym.vn/blog/2018/08/07/tho-lanh-nghe-9-nhung-thread-nguy-hiem-dich-vu-socket-4/)

**Bài trước:** [Thợ lành nghề #7: Một lần không đủ (Dịch vụ Socket 2)](https://codegym.vn/blog/2018/08/05/tho-lanh-nghe-7-mot-lan-khong-du-dich-vu-socket-2/)

**Tác giả:** Robert C. Martin

**Người dịch:** Hoàng Ngọc Diêu (conmale)