Avaliação da Confiabilidade de um Software utilizando Aspectos

Daniel P. Porto[1], Lylian F. de Souza[1], Mauricio Daniel R. Martinez[1], Daniele S.B.M. Neto[2], Nicolas Anquetil[1], Káthia M. de Oliveira[1]

[1] Universidade Católica de Brasília (UCB)
Caixa Postal – 91.501-970 – Brasília – DF – Brasil

[2] Universidade de Brasília (UnB)
Brasília – DF – Brasil

{danielpporto,lyliann,mdaniel.martinez}@gmail.com,
{anquetil,kathia}@ucb.br

Resumo. *Para garantir que os sistemas de software atendem às expectativas de seus usuários foram criados modelos de qualidade de software que tentam medir características importantes como a confiabilidade. Para medir a confiabilidade de sistemas de software de maneira contínua, é necessário definir meios automáticos de coletar as métricas de qualidade. Quando o sistema considerado é grande e mal conhecido (sistema legado), essa tarefa pode se revelar problemática. Neste artigo apresentamos uma experiência bem sucedida de usar a programação orientada a aspectos para acompanhar a confiabilidade de um sistema web programado em Java/JSP. Resultados das primeiras medições são apresentados.*

Abstract. Abstract. *To ensure that software systems address the expectation of the users, different software quality models have been defined. These models measure characteristics such as the reliability of the software. To measure reliability continually, it is important to collect automatically the data. When the software is a legacy system, this task can be very difficult. This paper presents a well succeed experiment in using aspect oriented programming to assess the reliability of a web software system coded in Java/JSP. Preliminary results are presented.*

1. Introdução

Gerar um sistema de software que atende às expectativas de seus usuários é um problema complexo necessitando grandes investimentos. Num esforço para garantir o retorno desses investimentos, foram criados modelos de qualidade de software que tentem avaliar em que medida um software atende ou não às expectativas de seus usuários. Dentro dessas expectativas inclui-se a confiabilidade do sistema, que é sua capacidade em manter um nível de desempenho adequado dentro de condições de uso definidas.

Esse projeto visa avaliar de forma contínua e sistemática a confiabilidade de um sistema institucional (de uma universidade) com mais de 1000 usuários. O sistema

considerado é complexo (>75 mil linhas de código Java, uso de diferentes padrões de programação internet, ...), sua documentação está desatualizada e ele é pouco conhecido por seus mantenedores atuais. Essas características impuseram restrições sobre a realização do projeto: (i) como a avaliação da confiabilidade é para ser contínua, deveria ser realizada de forma automática; (ii) como o sistema é grande e mal conhecido, pretende-se interferir o menos possível no código existente.

A solução encontrada para atender aos requisitos mencionados foi a utilização da programação orientada a aspectos, que permite a implementação de código adicional para atender a objetivos específicos e a inserção desses aspectos no sistema de forma adaptável e transparente, sem que haja necessidade de alteração de qualquer linha de código do software original.

Nas próximas seções será apresentada a conceituação básica de confiabilidade (Seção 2) e de aspectos (Seção 3). A Seção 4 apresenta como foram coletadas as métricas de confiabilidade escolhidas, inclusive com uma descrição dos aspectos definidos. Na Seção 5 são apresentados os resultados de uma fase de coleta inicial. Finalmente, na Seção 6, são apresentados as conclusões e os trabalhos em andamento.

2. Confiabilidade de Software

Segundo a norma ISO/IEC 9126 (2003) confiabilidade de software é a "capacidade do produto de software de manter um nível de desempenho especificado, quando usado em condições especificadas". Em termos estatísticos, confiabilidade é definida como a probabilidade de operação livre de falhas de um software, em um ambiente especificado, durante um período de tempo especificado.

A ISO/IEC 9126 (ISO/IEC TR 9126-2 e 9126-3, 2002) especifica 26 métricas de confiabilidade, oito internas, que podem ser aplicadas a um produto de software não executável (como uma especificação ou código-fonte) e 18 externas, que são mensuráveis a partir da execução e do comportamento do sistema. Essas métricas foram analisadas buscando identificar quais poderiam ser coletadas automaticamente. Com esse critério foram eliminadas as métricas internas que implicavam em intervenção humana (por exemplo: Detecção de defeitos - Quantos defeitos foram detectados em revisão de produto). Das 18 métricas externas, sete foram selecionadas por serem possíveis de mensurar automaticamente (ver Tabela 1).

Tabela 1. Métricas de Confiabilidade Selecionadas

Métrica	Descrição	Fórmula
Densidade de defeito latente estimada	Quantos defeitos ainda existem que podem surgir como falhas futuras	X= abs(A1-A2)/B; A1=número de defeitos latentes estimados; A2=número de defeito encontrados; B=tamanho do sistema
Densidade de defeitos	Quantos defeitos foram detectados durante o período de teste definido	X=A/B; A=número de defeitos encontrados; B=tamanho do sistema
Disponibilidade	O quanto o sistema é disponível para o uso durante um período de tempo especificado?	X=To/(To+Tr); To=tempo total de operação; Tr=tempo total de reparo

Métrica	Descrição	Fórmula
Prevenção de indisponibilidade	Com que freqüência o sistema causa a indisponibilidade completa do ambiente de produção	X=1-A/B; A=número de indisponibilidades; B=número de falhas
Remoção de defeito	Quantos defeitos foram corrigidos	X=A1/A2; A1=número de defeitos corrigidos; A2=número de defeitos encontrados
Tempo médio de indisponibilidade	Qual é o tempo médio em que o sistema fica indisponível quando uma falha ocorre	X=T/N; T=tempo total de indisponibilidade; N=número de indisponibilidades
Tempo médio entre falhas (MTBF)	Com que freqüência o software falha em operação	X=T/A; T=tempo total de operação; A=número de falhas detectadas

Várias métricas contam o número de falhas ou de defeitos. Um defeito é um erro na programação do sistema. Os defeitos são percebidos quando o usuário detecta uma falha na execução do sistema (execução incorreta do sistema). Os defeitos só podem ser descobertos por análise cuidadosa do código. Normalmente as falhas são mais facilmente detectadas e servem de índice da presença de um defeito. Para estimar o número de defeitos latentes num sistema (A1, primeira métrica), pode-se usar um modelo estatístico como proposto por Fenton e Pfleeger (1997, capítulo 10). Esses modelos permitem gerar uma estimativa do número de defeitos no sistema a partir do número de falhas.

Um desses modelos, bem conhecido, é o Jelinski-Moranda (Fenton, Pfleeger, 1997, p.376; Musa et al. 1987), que trabalha a partir das seguintes hipóteses:

- todo reparo é perfeito no sentido que corrige o defeito e não cria novos defeitos;
- portanto, a taxa de falhas deve decrescer a cada reparo;
- todas as falhas têm a mesma taxa, ou seja, todas as falhas contribuem igualmente para o risco (taxa) de falha.

Essas hipóteses são longe de serem realistas, mas por ser o modelo estatístico mais simples, este foi usado nesta primeira fase do projeto.

3. Aspectos

Pela natureza das métricas consideradas, implementá-las implicaria em modificar todo o código do sistema. Por exemplo, para calcular o tempo médio entre falhas, deve-se ter um *log* das falhas que acontecem. Devido ao grande tamanho do sistema, à sua complexidade e à falta de conhecimento detalhado deste, não era viável implementar manualmente código de *log* em cada lugar onde erros podem ser detectados (ver métricas implantadas, Seção 4). A solução adotada foi usar aspectos: porções de código que implementam funcionalidades que não são do domínio da aplicação (ex.: *log*, persistência, segurança, integridade de dados, checagem de erros, entre outros (Laddad, 2002)) e podem ser "inseridos" em pontos específicos de um sistema.

A programação orientada a aspectos introduz a separação de interesses. Interesses podem ser definidos em termos de requisitos específicos que capturam a funcionalidade central de um módulo (interesse central - *core concerns*), ou requisitos que satisfaçam a vários objetivos do sistema e que atravessam múltiplos módulos (interesses transversais -

crosscutting concerns), como autenticação, *logging*, persistência de dados, etc. Aspectos são usados para implementar os interesses transversais. É possível implementar os aspectos de forma independente das funcionalidades (interesse central) de cada aplicação.

Existem várias ferramentas que implementam aspectos em diferentes linguagens de programação. Para a linguagem Java em particular, diferentes ferramentas já foram propostas, tais como (O'Regan, 2005; Kersten, 2004): AspectJ, JBoss AOP, Nanning, Aspectwerkz e Spring AOP. Um dos mais conhecidos é o AspectJ que tem como benefícios possuir uma sintaxe semelhante a Java e ser compatível com ferramentas de apoio a programação como Eclipse, JBuilder, NetBeans ou Emacs. Dessa forma, para avaliar a confiabilidade foi decidido usar o AspectJ.

4. Coleta das métricas com aspectos

As métricas de confiabilidade de software escolhidas (Seção 2) são métricas compostas de métricas mais simples (chamadas diretas). Como foi visto, é importante para o projeto que as métricas de confiabilidade escolhidas possam ser coletadas automaticamente. São listadas nesta seção todas as métricas diretas e como foram coletadas. Em alguns casos, a métrica é, na realidade, uma aproximação do que a norma ISO 9126 pede, para permitir que a coleta seja feita automaticamente:

- **Número de defeitos corrigidos**: Atualmente estimado manualmente, a equipe de manutenção do sistema informa quais defeitos detectados foram tratados (ver trabalhos em andamento, perspectivas de automatização, na Seção 6).
- **Número de defeitos encontrados**: Atualmente estimado manualmente (ver trabalhos em andamento, perspectivas de automatização, na Seção 6).
- **Número estimado de defeitos latentes**: Existem dois métodos de coleta: (i) pode ser calculado usando um modelo estatístico que estima o número de defeitos a partir do número de falhas (ver Seção 2). Mas esse modelo precisa de uma certa quantidade de dados (histórico), portanto, nesta fase inicial, a métrica (ii) foi calculada a partir do número de defeitos encontrados até um momento e do número de defeitos corrigidos (i.e.: composta das duas métricas precedentes).
- **Número de indisponibilidades**: Calculado a partir do arquivo de *log* do servidor de aplicação internet. A norma se refere a indisponibilidades do ambiente de produção. No caso considerado, esse ambiente foi limitado ao servidor de aplicação internet. Esse servidor é dedicado ao único sistema estudado.
- **Tamanho do sistema**: calculado a partir do número de linhas de *byte-code*. Foi escolhido contar as linhas de *byte-code* por serem menos dependentes dos diferentes estilos de programação em código fonte Java. A métrica tem a desvantagem de ser de interpretação mais difícil (a que corresponde uma linha de *byte-code*?). Está sendo estudado a possibilidade de usar um fator de conversão médio de X linhas de *byte-code* para Y linhas Java (aprox. 3,17 linhas *byte-code* por 1 linha Java) para ajudar nesta interpretação.

- **Tempo total de indisponibilidade**: Calculado a partir do arquivo de *log* do servidor de aplicação Internet (tempo entre uma parada do servidor de aplicação internet e uma reinicialização).
- **Tempo total de operação**: Calculado a partir das horas de início e fim de cada sessão de trabalho (hora do primeiro e último clique do mouse na aplicação internet).
- **Tempo total de reparo**: Estimado igual ao tempo total de indisponibilidade. A norma ISO 9126 recomenda considerar unicamente os reparos automáticos para essa métrica, mas o sistema não tem nenhum tipo de recuperação automática implantado em caso de interrupção de serviço (do sistema ou do servidor de aplicação internet), portanto essa recomendação foi descartada.

As falhas foram controladas a partir do horário de acontecimento de erros de execução (exceções Java). As falhas correspondentes à execução sem interrupção (sem exceção) mas produzindo um resultado errado não foram contabilizadas. Isso se deve ao fato que tais falhas são mais raras, em geral ela são mais fáceis de corrigir e após 3 anos de uso do sistema, a maioria já foi sanada. O tempo de operação foi controlado a partir dos horários de início e fim de cada sessão de trabalho. Esses dados são obtidos a partir da criação de dois aspectos:

- ***Log* dos cliques de *mouse***: Este aspecto faz o *logging* da hora, da sessão de trabalho e da classe Java executada por cada clique na aplicação. O código do aspecto é apresentado na Figura 1. Em resumo, a declaração de ponto de junção (`pointcut`) define onde o código do aspecto (adendo - *advice*) será inserido. Nesse caso, o aspecto é ligado a toda execução de um método `doGet` (método ativado por uma interação no *browser* internet) definida dentro do pacote `br.ucb.sigep` (pacote da aplicação) e o adendo será executado antes desta chamada. O adendo recupera a sessão de trabalho (contem o identificador da sessão de trabalho) e o nome do arquivo Java onde o método `doGet` é declarado. Alem dessas informações, a classe `Log` irá acrescentar o horário corrente.
- ***Log* das exceções**: Este aspecto faz o *logging* da hora, da sessão de trabalho e da localização de todo erro (exceção Java) acontecendo no sistema. A localização consiste na classe Java executada e o número da linha de código dentro do arquivo que define essa classe. O código do aspecto é apresentado na Figura 2. Em resumo, o aspecto é ligado a todo bloco `try/catch` no código, para qualquer tipo de exceção (`Throwable`) e o adendo será executado antes do bloco `catch` Java. O adendo recupera a sessão de trabalho (contem o identificador da sessão de trabalho) e a localização (arquivo e linha) do bloco `try/catch` e realiza o *logging* dessas informações junto com o horário corrente.

```
package aspectos;
import javax.servlet.http.*;
public aspect LogTempo
{
  Log log=Log.getInstance(); // Classe para logging em arquivo
  pointcut executaServlet(HttpServletRequest request,
                          HttpServletResponse response)
           : args(request,response)
           && within(br.ucb.sigep..*)
           && execution(protected void doGet(HttpServletRequest,
                                             HttpServletResponse));
  before(HttpServletRequest request,HttpServletResponse response)
  : executaServlet(request,response)
  { // inicio do advice
    StaticSession ss = new StaticSession(request);
    String arquivo
          = thisJoinPointStaticPart.getSourceLocation().getFileName();
    log.escreveTempo(ss.getSessionID(),ss.getIdUsuario(),arquivo);
  }
}
```

Figura 1. Código do aspecto para *log* de cada clique na aplicação.

```
package aspectos;
public aspect LogErro
{
  Log log = Log.getInstance(); // Classe para logging em arquivo
  pointcut erroHandler()
           : within(br.ucb.sigep..*) && handler(Throwable+);
  before(): erroHandler()
  {  // inicio do advice
     StaticSession ss = new StaticSession(null);
     int linha
          = thisJoinPointStaticPart.getSourceLocation().getLine();
     String arquivo
          = thisJoinPointStaticPart.getSourceLocation().getFileName();
     log.escreveErro(ss.getSessinoID(), ss.getIdUsuario(),
                     "localização: " + arquivo + linha);
  }
}
```

Figura 2. Código do aspecto para *log* de cada erro (exceção Java)

5. Análise dos Resultados

Essas métricas foram coletadas para um sistema acadêmico criado internamente por professores e alunos de engenharia de software e que se tornou um sistema institucional com mais de 1000 usuários. O sistema realiza o suporte à direção de pesquisa, monitoramento os projetos e fornecendo informações sobre a infra-estrutura de pesquisa: cadastro de projetos (atualmente com mais de 400 projetos de pesquisa cadastrados), julgamento dos projetos, acompanhamento financeiro e acompanhamento de resultados. A natureza acadêmica do sistema propiciou o desenvolvimento com diferentes tecnologias e participação de diferentes programadores ao longo do tempo.

Para validação do método de medição automática, foram coletados dados do dia 27 de outubro 2005 até 10 de fevereiro de 2006. Além de corresponder ao cronograma do projeto, esse é um período de pouca atividade na organização (recesso universitário de janeiro) que autorizava colocar em produção código experimental (aspectos, ver Seção 4). A Tabela 2 apresenta alguns dados resumidos sobre o sistema: o número de defeitos latentes estimados nesta tabela foi calculado a partir do modelo estatístico Jelinski-Moranda (ver Seção 2 e (Fenton, Pfleeger, 1997, p.376)), o tamanho do sistema permeneceu constante durante essa fase. A Tabela 3 dá os valores das métricas diretas para 8 períodos (quinzenas) da fase experimental: o número de defeitos latentes estimados nesta tabela foi calculado a partir dos números de defeitos detectados e corrigidos (ver discussão Seção 4). A Tabela 4 mostra os primeiros resultados das métricas de confiabilidade para os 8 períodos (quinzenas) da fase experimental.

Tabela 2. Alguns dados sobre a fase experimental de medição

Número de sessões de trabalho	565
Duração média de uma sessão	807 segundos ≈ 13'27"
Média de cliques por sessão	11
Tamanho do sistema	243.220 linhas *byte code* 76.806 linhas Java
Número de defeitos latentes estimado	70,8 ≈ 71 defeitos

Tabela 3. Resultado das medições diretas

período	27/10 06/11	07/11 20/11	21/11 04/12	05/12 18/12	19/12 01/01	02/01 15/01	16/01 29/01	30/01 10/02
numero sessão	121	171	116	61	7	6	6	77
média cliques	8,04	9,56	11,43	18,66	11,71	26,17	9,33	12,91
Número de falhas	14	6	13	3	0	1	0	9
sessões c/ falhas	2	5	8	3	0	1	0	4
defeitos corrigidos	3	0	0	1	0	0	0	0
defeitos detectados	1	5	4	2	0	1	0	4
defeitos estimados	0	0	4	6	6	6	7	7
duração média	8'03"	13'46"	13'43"	17'35"	6'01"	44'21"	5'05"	16'30"
tempo total de indisponibilidade	1h58' 34"	29'56"	10'31"	0'00"	0'00"	0'00"	0'00"	0'44"

Tabela 4. Resultados das métricas compostas

período	27/10 06/11	07/11 20/11	21/11 04/12	05/12 18/12	19/12 01/01	02/01 15/01	16/01 29/01	30/01 10/02
Densidade de defeito latente estimada ($\times 10^{-06}$)	4,11	1,64	4,11	2,06	3,29	2,88	3,70	2,06
Densidade de defeitos ($\times 10^{-06}$)	4,11	2,06	1,64	8,22	0,00	4,11	0,00	1,64

período	27/10 06/11	07/11 20/11	21/11 04/12	05/12 18/12	19/12 01/01	02/01 15/01	16/01 29/01	30/01 10/02
Disponibilidade	0,89	0,99	0,99	1,00	1,00	1,00	1,00	1,00
Prevenção de indisponibilidade	0,64	0,83	0,92	1,00	1,00	1,00	1,00	0,89
Remoção de defeito	0,43	N/A	0,00	N/A	N/A	N/A	N/A	0,00
Tempo médio de indisponibilidade	23'42"	29'56"	10'31"	N/A	N/A	N/A	N/A	0'44"
Tempo médio entre falhas	1h09' 36"	6h32' 23"	2h02' 24"	5h57' 39"	N/A	4h26' 06"	N/A	2h21' 07"

Ainda é cedo para tirar qualquer conclusão destes números. Apesar da fase de experimentação ter durado quase quatro meses, foi um período de pouca atividade (feriado e recesso de janeiro) não representativo (por exemplo, se teve 565 sessões de trabalho nesses três meses e meio, teve 330 no mês seguinte, incluindo o recesso do carnaval).

6. Conclusão e Trabalhos em Andamento

Neste artigo foi apresentado um trabalho de medição da confiabilidade de um sistema internet programado em Java/JSP. O projeto tem por objetivo implementar apenas métricas que poderiam ser coletadas automaticamente para permitir um acompanhamento contínuo da confiabilidade. Por causa da complexidade do sistema e da falta de entendimento detalhado dele por parte da equipe de manutenção, foi decidido realizar um mínimo de intervenção no sistema mesmo. Para isso foram usados aspectos que permitiram coletar dados para as métricas sem precisar modificar o código fonte do sistema. Os aspectos definem que código deve ser executado (*advice*) para a coleta dos dados e onde esse código deve ser inserido (ponto de corte) dentro do código fonte.

Uma primeira fase de validação do modelo e de coleta experimental das métricas já foi realizada e mostrou a efetividade da solução escolhida. A próxima etapa do projeto deve ser a criação de uma página internet de confiabilidade dentro do próprio sistema onde os patrocinadores e os usuários poderão acompanhar a evolução da confiabilidade do sistema. Para isso, as métricas que ainda são coletadas manualmente precisarão ser automatizadas:

- **Número de defeitos corrigidos**: Atualmente estimado manualmente. É previsto estimar a métrica a partir do sistema Bugzilla onde a equipe de manutenção do sistema já cadastra todos os defeitos que lhe são relatados pelos usuários e todos os defeitos corrigidos.
- **Número de defeitos encontrados**: Atualmente estimado manualmente. É previsto estimar a métrica a partir das linhas de código onde acontecem as falhas. Quando várias falhas (por exemplo em sessões diferentes) acontecem na mesma linha, elas correspondem ao mesmo defeito (nem sempre localizado nesta linha). De fato, essa estimativa dá um valor um pouco maior que o real número de defeitos já que alguns defeitos em uma linha produzem em cascata falhas em outros lugares. Foi estimado informalmente que a super-estimação é menor que 110%.

Outras etapas do projeto incluem a escolha de um modelo estatístico de predição dos defeitos mais realista que o atual (Jelinski-Moranda) e buscar meios de melhorar a confiabilidade do sistema a partir das informações fornecidas pelas métricas.

Referências

Fenton, N.E., Pfeleeger, S.L. (1997). *Software Metrics: A Rigorous & Practical Approach*. PWS Publishing Company, 20 park plaza, Boston, MA 02116-4324, USA, 2nd ed.

Laddad, R. (2002). I want my AOP!, part 1. JavaWorld. Disponível em: http://www.javaworld.com/javaworld/jw-01-2002/jw-0118-aspect.html (acesso 13/03/2006)

Musa, J. D., Iannino, A. and Okumoto K. (1987). *Software Reliability: Measurement, Prediction, Application*. McGraw-Hill, New York

O'Regan, G. (2004) Introduction to Aspect-Oriented Programming. On Java. Disponível em: http://www.onjava.com/pub/a/onjava/2004/01/14/aop.html (acesso 13/03/2006)

Kersten, M. (2005) AOP tools comparison, Part 1. IBM developer works. Disponível em: http://www-128.ibm.com/developerworks/java/library/j-aopwork1/ (acesso 13/03/2006)

NBR ISO/IEC 9126 (2003), *Engenharia de software - Qualidade de produto. Modelo de qualidade*.

ISO/IEC TR 9126-2 (2002), Software engineering – Product quality – External metrics.

ISO/IEC TR 9126-3 (2002), Software engineering – Product quality – Internal metrics.