

基于敏捷开发的构件化软件可靠性研究

王世安

(广东松山职业技术学院 计算机系, 广东 韶关 512126)

摘要: 通过对敏捷开发中基于构件的增量迭代开发过程的分析, 并结合缺陷纠正率和测试充分度提出了一种可靠性分析方法. 该方法将基于构件的敏捷开发软件项目划分成优先级不同的功能构件, 通过对每次迭代过程中各个构件的可靠性点估计, 获取迭代结束时软件系统的可靠性. 最后以实际例子说明了方法的应用.

关键词: 敏捷开发; 构件; 可靠性; 增量迭代; 软件测试

中图分类号: TP311.5

文献标识码: A

文章编号: 1000-7180(2011)03-0119-04

Research on Component-Based Software Reliability Based on Agile Development

WANG Shi-an

(Department of Computer, Guangdong Songshan Polytechnic College, Shaoguan 512126, China)

Abstract: The incremental iteration process based on component in agile development was analyzed, a method of reliability analysis was proposed which combining the collective rate of defects and adequacy of test cases. In the method, the project based on component in agile development was divided into different functional components, the reliability point of each component in per iteration was estimated, the reliability of software system was gained when iteration was end. Finally, the example of application for the method was explained.

Key words: agile development; component; reliability; incremental iteration; software testing

1 引言

敏捷软件过程是最适应软件开发全球化、个性化、快速化、高质量新格局的先进的软件开发模式. 敏捷开发方法避免了软件错误的集中爆发, 因为其开发过程是渐进的、增量迭代和适时调整的, 也就是说, 迭代开发方法使得软件的部分功能通过早期用户测试使先产生的错误早发现, 通过用户反馈得到及时解决.

基于构件的软件开发方法是在吸取了已有软件开发方法经验基础上的一种全新的、高效的开发方法, 采用可复用的软件构件进行开发, 能够有效地提高软件开发的质量和效率^[1-2].

将敏捷开发模式和基于构件的软件开发模式结合, 形成新型的软件开发模式, 极大地提高软件复用程度和开发效率^[3].

由于敏捷开发过程是一个增量开发、持续测试的过程, 没有独立的测试阶段. 使得传统的构件软件系统的可靠性模型, 对于基于敏捷开发的构件化软件系统也并不完全适用, 为此, 对敏捷开发中的增量迭代开发过程进行分析, 通过划分优先级不同的功能构件, 进而提出一种可靠性分析方法.

2 基于构件的增量迭代开发

2.1 敏捷开发与增量迭代

敏捷开发方法以逐渐增加功能的方式经常地交付系统, 获得用户反馈以促进软件产品最终质量的提高. 交付得越频繁, 一些隐藏的错误发现得越早, 最终产品的质量就越高. 敏捷开发方法尽早地、经常地从客户那里获得反馈, 以确认当前的工作和需要改正的错误. 敏捷方法认为代码是最好的文档, 认为当前可工作的软件才是用户真正希望看到的. 客户

见到真正的产品,才能对当前的工作有个准确、清晰的评价和反馈,加速需求的浮现.在敏捷方法中,工作软件才是真正衡量项目进度的标准,在开发实践中,客户有权决定哪些功能优先开发,哪些功能后开发.因而敏捷开发过程是一个功能增量迭代的过程,在一个开发周期的不同阶段都有不同版本的产品,这也是一种多版本实践,但不同于传统可靠性设计中多版本^[4].

迭代增量开发方法弱化了早期的需求规格说明和设计.强化开发过程中动态反馈和演化分析.

2.2 增量迭代中的构件抽取

在敏捷开发方法中,先发布版本的功能特征集是后续版本的子集,这些功能特征子集可以抽取为相应的功能构件,构件的大小是由这些功能的优先级,用户素材估计完成时间的累积工作量等共同决定的^[5],这部分工作必须在版本计划时确定下来.有了版本计划,才能确定当前版本的迭代次数,以及迭代计划中每次放入的用户素材.

图 1 是基于构件的敏捷开发实践迭代增长过程示意图.在第一个版本中完成构件 1,在第二个版本中修改构件 1 中出现的错误,并增加新的构件 2.当完成第 k 个版本后,软件产品已经包含了 k 个构件.一般地,在开发过程中认为从构件 l 到构件 k ,其重要性和使用频率逐渐降低.

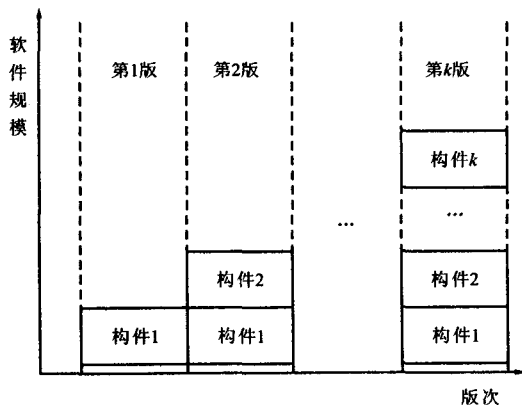


图 1 基于构件的增量迭代开发过程

基于构件的敏捷开发中的增量迭代开发过程,是和小版本发布实践相适应的.每一次迭代过后,在新的迭代中有一次用户需求增量;在每完成一个版本计划后,在新的版本中有一次增量.每个增量都是一些用户素材的集合,形成相应构件.开发人员和客户通过版本计划决定当前版本的范围,通过迭代计划决定每次迭代的范围.在结束第一个版本计划后,一个包含若干构件的最小工作软件完成,随后经过多次连续的增量,直到完成所有的用户需求,形成一

个完整的软件产品.

3 可靠性分析

3.1 可靠性分析基础

软件可靠性是指软件在规定的时间内软件不失效的概率.构件可以自行开发,可以购买商业构件,也可以从相关领域构件库中获取.虽然获取的构件一般都有相应的可靠性说明,但同一构件在不同的软件系统中所处状态不同,其可靠性也需要进行重新分析.在基于敏捷开发的构件化软件实践中,每发布一次软件版本,都是以构件测试为基础,用户必须对各构件进行使用性测试以发现软件失效.

小规模、多版本发布,使用户可以在早期发现软件中与自己所提的需求不一致的地方,用户可以持续地参与到测试中来.用户通过现场使用并快速反馈,为敏捷团队修正开发方向提供依据,可以说迭代增量开发、持续集成、小版本发布这些实践都是自适应的.对当前版本进行现场使用测试,获得失效数据(测试成功与失败次数),可获得该版本可靠性的点估计^[6].

在基于敏捷开发的构件化软件实践中,有两个影响敏捷项目软件可靠性增长的重要因素需要考虑:一是纠错率,表示了开发团队的纠错能力,影响可靠性的增长进度,用 p 表示;另一个是测试充分性,这体现测试用例的查错能力,用 μ 表示,令 $\mu \in (0, 1)$. μ 取值越接近 1,表示测试越充分.显然,如果测试不够充分,测出的可靠性值会比实际要高.

3.2 可靠性模型

用 $C_i (i = 1, 2, \dots, k)$ 表示不同的构件,用 $R_k^{(i)}$ 表示不同构件 C_i 在第 k 个版本中测得的可靠性点估计.这里采用简单的点估计方法,是因为测试成功与失败次数很容易在用户使用的阶段收集,关于失效时间的数据则较难收集.当然,也可以考虑在代码中嵌入系统日志来记录失效时间^[7],但是这样会占用大量系统资源.

一般地,同一个项目中,可以认为开发团队对软件的纠错能力保持不变.考虑测试充分性因子 μ 的影响,则 $\mu R_k^{(i)}$ 表示可靠性修正值.

在第二个版中 C_1 的预计可靠性为:

$$R_2^{(1)} = \mu R_1^{(1)} + (1 - \mu R_1^{(1)})p \quad (1)$$

上式可变为:

$$R_2^{(1)} = 1 - (1 - \mu R_1^{(1)})(1 - p) \quad (2)$$

同理可得,第 k 个版本时构件 C_1 的可靠性为

$$R_k^{(1)} = 1 - (1 - \mu R_1^{(1)})(1 - p)^{k-1} \quad (3)$$

因此通过式(3)可以预计构件 C_1 在迭代结束时可以达到的可靠性水平。

依次类推,得到第 i 个构件 C_i 在第 k 个版本时的可靠性预计值为

$$R_k^{(i)} = 1 - (1 - \mu R_1^{(i)})(1 - p)^{k-i} \quad (4)$$

在敏捷实践中,首先完成的构件和后续完成的构件没有直接依赖关系,但是在新版本中增加的构件,可能要用到上一版本中的构件的输出。如果说前一个版本的错误输出引起新构件失效,则说明前一个版本已经失效,这种失效只能算软件的一次失效。因此可以认为 k 个构件是独立的。图2显示不同版本增加的构件输入输出关系。

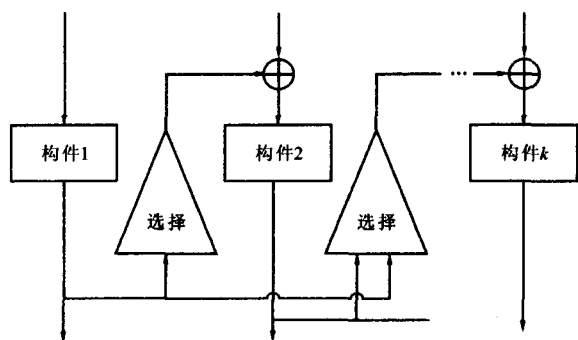


图2 各构件的输入输出情况

考虑到不同构件的规模、使用频率和重要性不一样,因而在测试用例设计上对不同构件有所偏重,不同构件的测试覆盖准则和测试用例数会不同。

用 $N_i (i = 1, 2, \dots, k)$ 表示不同构件测试时的测试用例数,令

$$\omega_i = \frac{N_i}{\sum_{i=1}^k N_i} \quad (5)$$

为可靠性权重,则在 k 个版本结束时,软件的可靠性可以经过递推得到。

用 f_i 表示使用 N_i 个测试用例对 C_i 进行测试得到的失效数,则系统的可靠性点估计为

$$R = 1 - \frac{\sum_{i=1}^k f_i}{\sum_{i=1}^k N_i} = 1 - \left[\frac{f_1}{\sum_{i=1}^k N_i} + \dots + \frac{f_k}{\sum_{i=1}^k N_i} \right] \quad (6)$$

设 f_i^* 为第 i 个构件第一次使用测试时发现的失效数,可以认为 f_i 是在 f_i^* 的基础上,以纠错率 p 经过 $k-i$ 次修改后剩下的失效数。即:

$$f_i = f_i^* (1 - p)^{k-i} \quad (7)$$

由上式可以得到

$$\frac{f_i}{N_i} = \frac{f_i^*}{N_i} (1 - p)^{k-i} = (1 - \mu R_1^{(i)})(1 - p)^{k-i} \quad (8)$$

$$f_i = N_i (1 - \mu R_1^{(i)})(1 - p)^{k-i} \quad (9)$$

带入式(4),有

$$R_i = 1 - \sum_{i=1}^k \omega_i (1 - \mu R_1^{(i)})(1 - p)^{k-i} \quad (10)$$

由此得到

$$R = 1 - \sum_{i=1}^k \omega_i (1 - R_k^{(i)}) \quad (11)$$

通过式(11)可以计算得到迭代结束时软件的可靠性。

4 实际应用

某研究所飞行器型号软件项目采用构件化敏捷模式开发,该项目划分了5个优先级不同的功能构件,经过多次迭代集成了产品需求,产生5个版本,并基于客户使用剖面在客户现场对每个版本的新增构件进行了可靠性测试(其余为纠错测试),测试结果如表1所示。

表1 可靠性第1次测试结果($\mu = 0.95$)

构件	N_i	f_i^*	$R_1^{(i)}$	$\mu R_1^{(i)}$	ω_i
1	183	31	0.8306	0.7891	0.2221
2	172	28	0.8372	0.7953	0.2087
3	165	29	0.8242	0.7830	0.2002
4	156	27	0.8269	0.7856	0.1893
5	148	22	0.8514	0.8088	0.1796

根据表1的测试数据,计算各版本阶段不同构件的可靠性演变见表2。

表2 可靠性演变情况($p = 0.5, \mu = 0.95$)

版次	C_1	C_2	C_3	C_4	C_5	R
1	0.7891					
2	0.8946	0.7953				
3	0.9473	0.8977	0.7830			
4	0.9736	0.9488	0.8915	0.7856		
5	0.9868	0.9744	0.9458	0.8928	0.8088	0.9262

表2中,使用公式(4)计算出构件 $C_1 - C_5$ 在最后一次迭代结束的可靠性预计。 C_5 是最后加入的构件,第一次测得的可靠性点估计为0.7891。从表2看出,软件的核心功能如 C_1 、 C_2 在迭代结束时已达到较高的可靠性,但对整个软件而言,由于受后面增加的功能组的影响,此时可靠性很低,还需要进一步测试。在后续测试过程中,如果受上市时间的限制,或客户要求工期的限制,可以暂时关闭次要功能(如 C_4 、 C_5)的使用,以获得主要功能较高的可靠性,这符合敏捷实践的思想。使用公式(11)计算出迭代结束时软件可靠性为0.9262。若要进一步提高软件可靠性,可以根据不同构件的可靠性要求,继续进行可靠性测试。

实际应用也说明开发人员可以在版本计划阶段考虑不同构件的可靠性分配。

5 结束语

基于构件的敏捷软件开发过程是一个增量开发、持续测试的过程,没有独立的测试阶段且受构件固有特性的影响,传统构件软件系统可靠性模型目前还无法使用到敏捷开发过程中。因此,开发基于敏捷开发过程和测试实践的新的构件化软件可靠性估计模型是有一定价值的。敏捷实践是测试驱动的开发方法,在开发过程中测试用例是不断调整的,测试用例揭示错误的能力和改错能力是随着开发活动提升的,所以应考虑测试充分度随开发周期的变化,这将在后续工作中考虑这一变化。

参考文献:

- [1] 周国强,黄达明,曾庆凯. 基于安全属性分解的构件安全性量化和计算[J]. 微电子学与计算机,2008(10): 43—46.
- [2] 张驰. 软件构件匹配技术研究[J]. 微电子学与计算机,2009(3): 109—112.
- [3] 潘悦,沈备军. 基于构件的敏捷软件开发方法[J]. 计算

机工程,2005,31(15):68—70.

- [4] Yi Ding, Nannette Napier. Measurement framework for assessing risks in component—based software development [C] // Proceedings of the 39th Hawaii International Conference on System Sciences. Washington D C: IEEE Computer Society, 2006.
- [5] Arie van Deursen, Customer involvement in extreme programming[J]. Software Engineering Notes, 2001, 26(6): 70—73.
- [6] Kuppuswami S, Vivekanandan K, Ramaswamy P, et al. The effects of individual XP practices on software development effort[J]. ACM SIGSOFT Software Engineering Notes, 2003, 28(6): 1—6.
- [7] Nancy Van S, Ron Morssicarto. Taming the Embedded Tiger—Agile test techniques for Embedded software [C] // Proceedings of the Agile Development Conference (ADC'04). Washington, DC, USA: IEEE Computer society, 2004: 120—126.

作者简介:

王世安 男,(1971—),硕士,副教授,高级工程师。研究方向为算法设计与分析、计算机应用与软件。

(上接第 118 页)

传感器节点数据进行打包压缩,以期获得更好的查询效果。

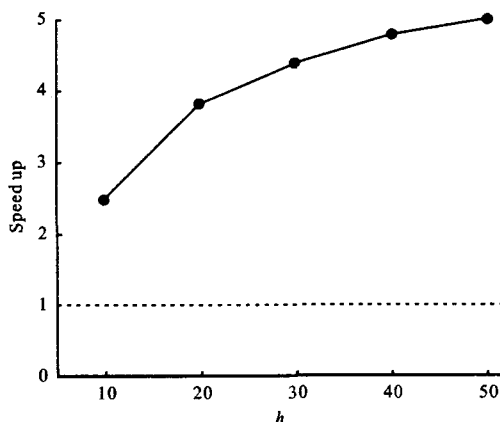


图 8 h 对加速比的影响

参考文献:

- [1] Sylvia Ratnasamy, Brad Karp. GHT: A geographic hash table for data centric storage [C] // The 1st ACM Int'l Workshop on Wireless Sensor Networks and Applications (WSNA 2002), Atlanta, USA: ACM, 2002.
- [2] Deepak Ganesan, Deborah Estrin, John Heidemann. Dimensions: why do we need a new data handling architecture for sensor networks[J]. ACM Sigcomm Com-

puter Communication Review, 2003, 33(1): 143—148.

- [3] A Deshpande, C Guestrin, W Hong, et al. Exploiting correlated attributes in acquisitional query processing [C] // IEEE Int'l Conf on Data Engineering (ICDE), Tokyo, Japan: IEEE, 2005.
- [4] Peter Desnoyers, Deepak Ganesan, Prashant Shenoy. TSAR: A two tier sensor storage architecture using interval skip graphs [C] // ACM Conf on Embedded Networked Sensor Systems (SENSYS'05), San Diego, USA: ACU, 2005.
- [5] Raghavendra C S, Krishna M Sivalingam, Taieb Zhati, et al. Wireless Sensor Networks [M]. Berlin: Kluwer Academic Publishers, 2004: 185—252.
- [6] 亢晓丽,王换招,张鹏. EERPLS: 一个大规模传感器网络中的高能路由协议[J]. 微电子学与计算机, 2009(5): 242—247.

作者简介:

胡 华 男,(1964—),教授,博士。研究方向为移动计算、软件工程。

苗媛媛 女,(1984—),硕士。研究方向为无线传感网络的数据管理。

庄 毅 男,(1978—),博士,副教授。研究方向为数据管理。