Gerd Streckfuss, IQM

QFD in der Softwareentwicklung: Deployment-Struktur mit Test-HoQ zur Reduzierung des Testaufwandes bei der Softwareerstellung

1. Einleitung

Viele Untersuchungen über die Kostenstruktur einer Softwareentwicklung haben folgende Ergebnisse gezeigt:

- ☞ Es wurde wenig Aufwand für die Ermittlung der Markt-/Kundenbedürfnisse getrieben.
- ☞ Die Kosten für die Programmierung (im wesentlichen für die Erstellung des Quellcodes) betrugen selten mehr als 30% der Gesamtentwicklungskosten.
- ☞ Die Kosten der verschiedenen Tests (Verifikation und Validierung) mit den notwendigen Änderungen und Nachtests lagen immer über 50%, meist um die 70% der Gesamtkosten und sie waren *der* Kostentreiber in der Softwareentwicklung.
- ☞ Allein der Anteil Fehlerbeseitigungszeiten an der gesamten Programmierzeit wurde mit 35 - 50% ermittelt.

Auch wenn durch moderne Testroutinen, Testsoftware u.s.w. die Qualität des Outputs in den letzten Jahren erhöht wurde, ist der Testaufwand anteilsmäßig an den Gesamtkosten nicht deutlich reduziert worden. Dies liegt natürlich auch daran, dass die Systeme insgesamt wesentlich komplexer wurden.

Auch können die Fehlerfindungskosten - wie in der Vergangenheit üblich - , nicht mehr ohne weiteres auf den Anwender/Kunden abgewälzt werden (Betatests). Der Kunde verlangt zunehmend fehlerfreie Software schon im ersten Release.

2. Deploymentstruktur für die Planung von Softwaretests.

Anfang der neunziger Jahre wurde bei Schlumberger Technologies ein QFD Deployment entwickelt mit dem Ziel:

- ☞ die Effektivität der Tests zu erhöhen,
- ☞ den Testaufwand zu reduzieren und
- ☞ die verbleibenden Fehler zu reduzieren.

Die Struktur dieser SW-Test-QFD wurde in den letzten Jahren verfeinert und an verschiedene Applikationen angepasst.

2.1 Vorgehensweise

Der zeitliche Ablauf kann in folgende Schritte (die meist einem HoQ entsprechen) unterteilt werden:

Schritt 1:

Es wird ein HoQ1 erstellt, wie es auch in der "klassischen" QFD Anwendung der mechanischen Produktentwicklung eingesetzt wird.

Es hat sich dabei gezeigt, dass der "normale" Anwender bei den Anforderungen keine Differenzierung zwischen einer Applikationssoftware, dem Betriebssystem, der Hardware und der Peripherie vornimmt.

Schritt 2:

Diese Unterscheidungen werden anschließend von den Fachabteilungen vorgenommen. Dabei werden, um den Aufwand möglichst gering zu halten, nur die neu zu erstellenden oder größere Änderungen betreffenden Module berücksichtigt.

Schritt 3:

Die Anforderungen an die Codierung sind üblicherweise in User-Requirement-Katalogen oder in Pflichtenheften beschrieben. Es gibt hier keine Matrix, da es bisher noch nicht gelungen ist, die Codierung als Umsetzungsmatrix zu definieren.

Schritt 4:

In dieser Phase werden die einzelnen, begrenzten Testmöglichkeiten, die der Entwickler selbst sofort einsetzen kann, aufgelistet und mit den neuen Features (Modul, Tasks u.s.w) abgeglichen. Die Beziehungsfrage bei der Erstellung der Matrix heißt: Wie und wieviel Fehlerfreiheit kann diese Testroutine für das neue Feature gewährleisten.

Ziel ist es, dabei festzustellen, ob alle neuen/geänderten Module durch den Test erfasst werden. Falls nicht (= leere Zeilen), muss dieser neu definiert werden. Umgekehrt können nicht benötigte Testroutinen wegfallen (= leere Spalten). Diese Tests sind Einzeltests, meist auf Modulbasis, also noch keine System(Intergrations-)tests.

Im Dach des jeweiligen HoQ können, falls notwendig, die Korrelationen der Testroutinen untereinander ermittelt werden, um:

- ☞ Konflikte der Routinen untereinander zu erkennen und um
- ☞ Routinen mit gemeinsamen/überlappenden Möglichkeiten zu ermitteln.

Danach kann der gesamte Testvorgang optimiert werden.

Schritt 5:

Der wichtigste und meist auch aufwendigste Test ist der Systemtest, bei dem das Zusammenspiel aller Komponenten getestet wird. Hier erst kann die Qualität des Outputs getestet werden.

Normalerweise sind schon eine Menge von System-Testroutinen vorhanden, die aus der vorherigen Versionen stammen.

Unbekannt ist aber in der Planungsphase, ob diese Tests auch die neuen bzw. geänderten Module richtig erfassen. Dies wird im System-Test-HoQ ermittelt.

Dadurch wird sichergestellt, dass alle Änderungen getestet werden, unnötige Tests entfallen und für zu-

künftige Tests ein solides Testprogramm zu Verfügung steht. Die Praxis hat gezeigt, dass bei Fehlern, die trotzdem beim Kunden noch auftreten, diese meistens minimale Auswirkungen auf die Performance der Anwendung haben. Werden die Fehler analysiert und die dazugehörigen Testroutinen verbessert, erhält man für zukünftige Tests einen robusten Testablauf.
Die Matrix enthält auch noch weitere Informationen wie Testaufwand, Testvoraussetzungen, Testergebnisse u.s.w.

3. Fazit.
Solange der Kunde über den Updatepreis die Fehlerkosten übernimmt, ist Testen zweitrangig. Falls dies nicht mehr der Fall sein sollte, müssen die Testkosten drastisch reduziert werden, da der Kunde - und dies ist in der "mechanischen" Welt schon eingetreten - nicht mehr bereit ist, die Fehlerkosten zu übernehmen. Die vorbeugende Fehlervermeidung spielt dann ein große Rolle, und dies kann mit dem vorgestellten SW-QFD-Deployment erreicht werden.