KONTINUIERLICHES QUALITÄTS-CONTROLLING: MITTEL GEGEN DEN QUALITÄTSVERFALL IN DER SOFTWAREWARTUNG

Spätestens seitdem Manny Lehman Mitte der 70er Jahre seine Gesetze der Softwareevolution formuliert hat, wissen wir, dass Softwaresysteme einem schleichenden Qualitätsverfall unterliegen. Häufig bleibt unerwähnt, dass diese Gesetze nur gelten, wenn keine geeigneten Gegenmaßnahmen ergriffen werden. Dieser Artikel beschreibt, wie kontinuierliches Qualitäts-Controlling hilft, dem Qualitätsverfall entgegenzuwirken und dadurch die Qualität von Softwaresystemen langfristig sicherzustellen.

die autoren

Dr. Florian Deißenböck
(E-Mail: deissenboeck@cqse.eu)
ist geschäftsführender Gesellschafter der CQSE GmbH. Zu seinen Interessen gehören die Modellierung von Qualitätszielen und die Wirtschaftlichkeitsbetrachtung von langfristig angelegten Qualitätsinitiativen.

Dr. Benjamin Hummel
(E-Mail: hummel@cqse.eu)
ist Mitbegründer der CQSE GmbH. Seine Schwerpunkte liegen in der Beratung zur Einführung sowie Umsetzung von flächendeckenden Peer-Reviews und der Werkzeugunterstützung für kontinuierliches Qualitäts-Controlling.

Der Stellenwert von Softwarequalität ist unter Entwicklern und Entscheidern meist unumstritten. Kurzfristig ist eine hohe Qualität wichtig, um den Anwender optimal zu unterstützen. Langfristig sind ein optimaler Betrieb und eine kostengünstige Weiterentwicklung nur möglich wenn die Software entsprechenden Qualitätsansprüchen genügt.

Die Norm ISO 9126 (vgl. [ISO03]) (bzw. deren Nachfolger ISO 25010) unterteilt die Produktqualität von Software in die folgenden Bereiche:

- Funktionalität
- Zuverlässigkeit
- Nutzbarkeit
- Effizienz
- Wartbarkeit
- Portierbarkeit

Für die Funktionalität gibt es mit den verschiedenen Spielarten des Softwaretests vielfältige Lösungsansätze, die in Form von Lasttests und Nutzertests teilweise auch für Zuverlässigkeit, Nutzbarkeit und Effizienz Anwendung finden. Die Bereiche Wartbarkeit und Portierbarkeit werden in der Praxis jedoch oft nur sehr oberflächlich betrachtet. Das verwundert, da gerade diese beiden Faktoren einen wesentlichen Einfluss auf die langfristigen Kosten von Betrieb und Weiterentwicklung eines Softwaresystems haben, die typischerweise ein Vielfaches der initialen Entwicklungskosten betragen.

Der in der Praxis am häufigsten anzutreffende Ansatz, für eine hohe Produktqualität zu sorgen, besteht aus einer Verbesserung der Prozessqualität, etwa nach dem *Capability Maturity Model (CMM)* (vgl. [Pau95]). Allerdings zeigt sich immer wieder, dass der Prozess zwar einen Einfluss auf das Produkt hat, man aber auch mit einem sehr „reifen" Entwicklungsprozess (unabhängig davon, ob agil oder nicht) qualitativ schlechte Software produzieren kann und umgekehrt (vgl. [Voa97]). Die offensichtliche Lösung besteht darin, nicht nur die Entwicklungsprozesse, sondern das Produkt selbst zu beobachten und zu verbessern – im Fall von Software also den Quellcode sowie gegebenenfalls Dokumente und Modelle. Genau dieses Ziel wird beim Qualitäts-Controlling verfolgt, das den größten Nutzen entfaltet, wenn es nicht sporadisch oder gar nur einmalig, sondern kontinuierlich angewandt wird. Nur so lassen sich Defizite frühzeitig erkennen, wenn ihre Beseitigung noch einfach ist und eine schleichende Verschlechterung der Qualität verhindern.

Umsetzung des Qualitäts-Controllings

Kontinuierliches Qualitäts-Controlling lässt sich vereinfacht als eine Regelungsschleife auffassen (siehe Abbildung 1). Dabei unterliegt das System ständigen externen Einflüssen, wie etwa neuen Anforderungen und Änderungen durch die Entwickler. Gleichzeitig nimmt man an,

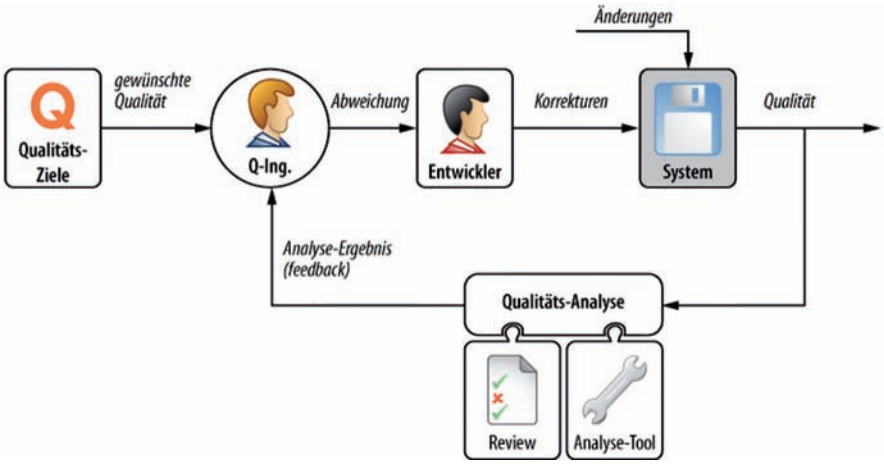

Abb. 1: Kontinuierliches Qualitäts-Controlling als Regelungsschleife.

$$171 - 3.42 \cdot \ln(avgHV) - 0.23 \cdot avgCC - 16.2 \cdot \ln(avgLOC) + 50 \cdot \sin(\sqrt{2.4 \cdot perCM})$$

HV: Halstead-Volumen CC: Zyklomatische Komplexität
LOC: Lines of Code perCM: % Kommentarzeilen

Abb. 2: Negatives Beispiel für eine Qualitätsmetrik: der Maintainability-Index.

dass das System eine bestimmte inhärente Qualität besitzt, die durch eine Qualitätsanalyse bestimmt wird. Ob die Qualität den vorgegebenen Qualitätszielen entspricht, wird durch die Rolle eines Qualitätsverantwortlichen überprüft, der dann regelnd auf die Entwickler einwirkt. Dabei kann die Rolle des Qualitätsverantwortlichen bei entsprechender Ausbildung durchaus auch von den Entwicklern selbst ausgefüllt werden. In großen Projekten mit komplexen Analyseszenarien kann eine Trennung von der Entwicklerrolle jedoch sinnvoll sein, um die Entwickler durch eine Vorfilterung und Interpretation der Analyseergebnisse zu entlasten. Diese Schleife durchläuft man regelmäßig wöchentlich, täglich, stündlich oder – im Idealfall – *kontinuierlich*. Wichtig sind in diesem Bild zwei Teile:

Zum einen sollte eine Vorstellung davon bestehen, was die Qualitätsziele sind, also was gute Software ausmacht. Hier gibt es keine absoluten Wahrheiten, sondern die Ziele hängen immer auch vom Projektkontext ab.

Der andere wesentliche Teil ist die Qualitätsanalyse, denn nur wenn ein möglichst verzerrungsfreies Bild der Systemqualität gezeichnet wird, kann man zielgerichtet eingreifen. Eine erfolgreiche Qualitätsanalyse besteht dabei immer aus zwei Komponenten: automatischen Analysen und manuellen Inspektionen. Die automatischen Analysen bieten den Vorteil, dass sie bei geringem Aufwand sehr schnell einen Überblick auch über sehr große Codemengen vermitteln. Da maschinelle Verfahren aber immer nur einen sehr beschränkten Ausschnitt von Qualitätsattributen überprüfen können, ist es wichtig, diese durch manuelle Verfahren zu ergänzen.

Bei der Auswahl der automatischen Analysen darf man nicht wahllos in den „Metrik-Baukasten" greifen, da viele Werkzeuge und verbreitete Metriken leider in erster Linie das messen, was einfach zu messen ist, und nicht das, was die Qualität ausschlaggebend beeinflusst. Ein Negativbeispiel hierfür ist der so genannte *Maintainability-Index* (vgl. [Oma92]), dessen Berechnung Abbildung 2 zeigt. Auch wenn die Autoren Schwellwerte für hohe oder niedrige Wartbarkeit angeben, bleibt unklar, wie diese Metrik interpretiert werden soll und welche Maßnahmen sich aus schlechten Werten ableiten. Ironischerweise wird gerade der Maintainability-Index in sehr vielen Werkzeugen bestimmt (so auch in „Visual Studio 2008"), während das Software Engineering Institute (SEI), an dem der Index entstanden ist, diese Metrik inzwischen selbst als „Legacy" bezeichnet (vgl. http://web.archive.org/web/20080509131004/http://www.sei.cmu.edu/str/descriptions/mitmpm.html) und die entsprechenden Informationen mittlerweile sogar von den Web-Seiten entfernt hat. Stattdessen sollten Analysen eingesetzt werden, die auf unmittelbare Probleme im Code hinweisen. Die Relevanz solcher Messungen ist für die verschiedenen Mitglieder des Entwicklungsteams deutlich nachvollziehbarer als abstrakte Metriken und die Ableitung konkreter Gegenmaßnahmen gestaltet sich wesentlich leichter. Ein Beispiel für eine solche Analyse ist die Erkennung von Code-Klonen, d. h. redundanten Code-Teilen, die durch Copy&Paste entstanden sind (vgl. [Jür09]). Abbildung 3 zeigt einen solchen Klon. Obwohl Klone die Test- und Änderungsaufwände erhöhen und das Risiko bergen, dass inkonsistente Änderungen von Klonen zu Fehlern führen, finden sich in der Praxis oft Systeme, deren Code zu über 20 % aus Klonen besteht.

Andere geeignete automatische Analysen sind:

- einfache Schwellwerte für die Code-Struktur (z. B. Methoden mit mehr als 40 Zeilen)
- die Suche nach Bug-Patterns

```
// Utilities for arrays of elements
public String showElements(ModelElement[] elements, String nomsg) {
    boolean found = false;
    StringBuffer res = new StringBuffer();
    if (elements != null) {
        Index.getInstance().setCurrentRenderer(
                FlatReferenceRenderer.getInstance());
        for (int i = 0; i < elements.length; i++) {
            ModelElement el = elements[i];
            res.append(showElementLink(el)).append(HTML.LINE_BREAK);
            found = true;
        }
        Index.getInstance().resetCurrentRenderer();
    }
    if (!found && nomsg != null && nomsg.length() > 0) {
        res.append(HTML.italics(nomsg));
    }
    return res.toString();
}
```

```
// Utilities for arrays of elements
public String showElements(ModelElement[] elements, String nomsg) {
    boolean found = false;
    StringBuffer res = new StringBuffer();
    if (elements != null) {
        Index.getInstance().setCurrentRenderer(
                FlatReferenceRenderer.getInstance());
        for (int i = 0; i < elements.length; i++) {
            ModelElement el = elements[i];
            res.append(showElementLink(el)).append(HTML.LINE_BREAK);
            found = true;
        }
        Index.getInstance().resetCurrentRenderer();
    }
    if (!found && nomsg.length() > 0) {
        res.append(HTML.italics(nomsg));
    }
    return res.toString();
}
```

Abb. 3: Beispiel für einen Code-Klon mit einer Inkonsistenz.

- die Identifikation von Abweichungen zwischen Soll- und Ist-Architektur

Für den Einsatz all dieser Verfahren ist es sehr wichtig, die Werkzeuge an den Entwicklungskontext anzupassen. So ist es beispielsweise irrelevant, wenn automatisch generierter Code übermäßig lange Methoden enthält, sofern dieser Code nicht von Hand geändert wird. Zudem sollte man durch Anpassungen die Rate der *False Positives* in den niedrigen einstelligen Prozentbereich drücken, da sonst eine hohe Zahl von Falschmeldungen Ressourcen unnötig bindet und die Akzeptanz durch die Entwickler verhindert.

Trotz der vielfältigen Entwicklungen im Bereich der automatischen Analysen können diese nur einen Ausschnitt der relevanten Qualitätskriterien überprüfen. Auch wenn es verlockend erscheint, sich auf den automatisierbaren Teil zu beschränken, zeigt sich immer wieder, dass die tiefer liegenden Probleme oft nur durch manuelle Inspektionen gefunden werden können. Die Kreativität und Erfahrung der Entwickler ist dabei der starren Struktur der Analyseprogramme überlegen. Bei der Art der Inspektionen ist es weniger wichtig, ob es sich um eine Inspektion während der Entstehung und Änderung des Codes (*Pair-Programming*) handelt oder um einen nachgelagerten leichtgewichtigen Review-Prozess (*Peer-Review*). Wichtig ist aber, dass die Ergebnisse der automatischen Analysen während der Inspektion zur Verfügung stehen, da diese die manuellen Teile ergänzen. Gerade die globale Sicht auf den Code (etwa für Klone) und stupide systematische Prüfungen (Zeilen zählen) werden von Menschen nur ungenau durchgeführt, während gerade diese Aspekte sehr einfach zu automatisieren sind. Gleichzeitig sollten Informationen aus diesen manuellen Analysen, wie etwa die Review-Ergebnisse, auch wieder in die automatischen Analysen zurückfließen, was mit entsprechender Werkzeugunterstützung kein Problem ist.

Die Operationalisierung des kontinuierlichen Qualitäts-Controllings findet meist in Form von so genannten *Quality Dashboards* und Qualitätsberichten statt. Ein Dashboard liefert dabei eine aggregierte Sicht auf den aktuellen Qualitätsstand des Systems in Form von Ampelbewertungen, Übersichtsgrafiken (etwa in Form von *Tree-Maps*) oder Trendbetrachtungen für verschiedene Qualitätsaspekte. Beispiele für solche Visualisierungen, wie sie etwa das Open-Source-Werkzeug „ConQAT" (vgl. [Dei08]) anbietet, zeigt Abbildung 4. Das Dashboard wird dabei täglich oder stündlich automatisch aktualisiert und bietet so stets einen aktuellen Überblick über den aktuellen Qualitätsstand des Systems. Somit unterstützt es die Steuerung von Aktivitäten der Qualitätssicherung im Tagesgeschäft.

Einen eher strategischen Charakter haben dagegen Qualitätsberichte. Diese werden typischerweise mit größerem zeitlichem Abstand erstellt (etwa alle ein oder zwei Monate) oder beim Erreichen bestimmter Meilensteine (z. B. Zwischen-Releases). Ein Qualitätsbericht verbindet eine Momentaufnahme der Qualität des Softwaresystems mit einer Interpretation des Ist-Stands und einer Planung und Prognose der weiteren Qualitätsentwicklung. Der Qualitätsbericht hat zum einen dokumentierenden Charakter, zum anderen ist er durch den Interpretationsteil als Kommunikationsmittel zu Management und Auftraggeber geeignet. Die Erstellung eines solchen Berichts ist bei geeigneter Werkzeugunterstützung mit wenig Aufwand verbunden, da viele Inhalte direkt aus den kontinuierlichen Messungen übernommen werden können. Zu ergänzen sind dann lediglich die interpretierenden Abschnitte. Für diese Anteile kann es jedoch hilfreich sein, auf externen Sachverstand zurückzugreifen, weil die Projektbeteiligten häufig eine gewisse Betriebsblindheit gegenüber Qualitätsdefiziten des eigenen Systems entwickeln.

Qualitäts-Controlling in der Praxis

Für den Einsatz eines Qualitäts-Controllings gibt es zwei wesentliche Szenarien, die sich in einigen Aspekten voneinander unterscheiden:

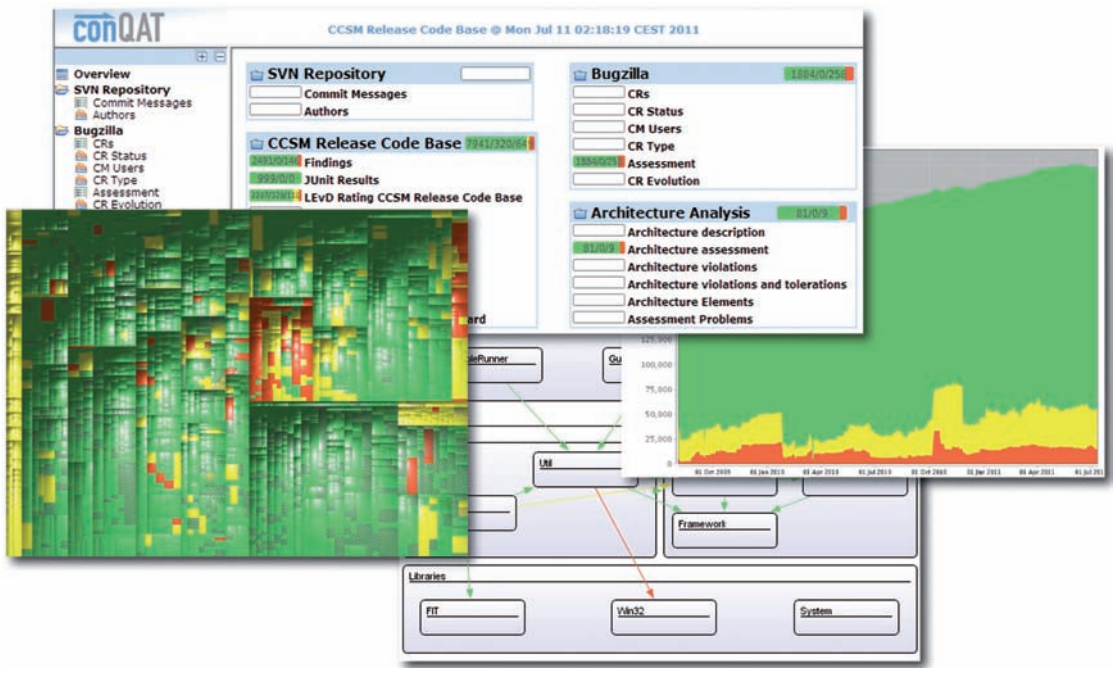

Abb. 4: Verschiedene Visualisierungen in einem Quality Dashboard.

- Einmal kann ein Qualitäts-Controlling unternehmensintern genutzt werden um die Qualität des eigenen Entwicklungs- oder Wartungsprojekts im Griff zu behalten. Die Motivation für die Qualitätssteuerung kann aus dem Entwicklungsteam selbst kommen oder durch Vorgaben der Führungsebene oder des Auftraggebers. Charakteristisch hierbei ist, dass Entwicklung und Controlling beide innerhalb des gleichen Unternehmens stattfinden.
- Das andere Szenario ist die Qualitätskontrolle eines Zulieferers. Hier führt der Auftraggeber das Qualitäts-Controlling durch, während die Entwicklung davon getrennt vom Zulieferer durchgeführt wird. In diesem Fall ist eine gewisse Qualität meist vertraglich vorgegeben und die Ergebnisse einer Qualitätsbeobachtung – etwa in Form von Qualitätsberichten – können Teil der Abnahmekriterien sein. Hierbei ist es wichtig, dass bereits zum Vertragsschluss definiert wird, was Qualität ist, und dass dies in Form von objektiven und messbaren Indikatoren festgehalten wird.

Im Gegenzug zu einer In-House-Entwicklung, bei der das Controlling in erster Linie der eigenen Verbesserung dient, ist in einer Zulieferer-Situation eine spätere Änderung oder Ergänzung der Maßzahlen nur schwer möglich. Entsprechend ist es bereits bei Festlegung der Verträge wichtig, ein gutes Verständnis der relevanten Qualitätskriterien zu haben. Das schrittweise Herantasten und Lernen kann sich hier später rächen.

Ein weiterer entscheidender Aspekt bei der praktischen Umsetzung von kontinuierlichem Qualitäts-Controlling ist der Umgang mit Altlasten. Nur wenige Projekte haben den Luxus, bereits von der ersten Stunde an ein Qualitäts-Controlling installiert zu haben. Viel häufiger hat man die Situation, dass in einem sehr umfangreichen System – oft bereits mit deutlichen Qualitätsdefiziten – ein Qualitäts-Controlling eingeführt wird. Das Ziel ist dabei, zumindest bei den neu entwickelten und geänderten Bestandteilen eine hohe Qualität zu sichern und in kleinen Schritten auch die bestehenden Defizite zu adressieren. Problematisch ist es hierbei, dass die Änderungen im Vergleich zum bestehenden System oft verschwindend klein sind. In einer stark aggregierten oder akkumulierten Sicht sind die Auswirkungen, die die Änderungen auf die Qualität haben, somit nicht sichtbar. Eine etablierte Lösung für dieses Problem ist die Definition einer *Baseline*. Diese fixiert den Ist-Stand, der bei Einführung des Qualitäts-Controllings angetroffen wird. Auf Basis der Baseline lassen sich Sichten auf die Qualitätszahlen erstellen, die nur die neuen und geänderten Teile umfassen. Bei der Werkzeugauswahl sollte dieser Aspekt berücksichtigt werden, da viele verbreitete Tools hier noch Nachholbedarf haben.

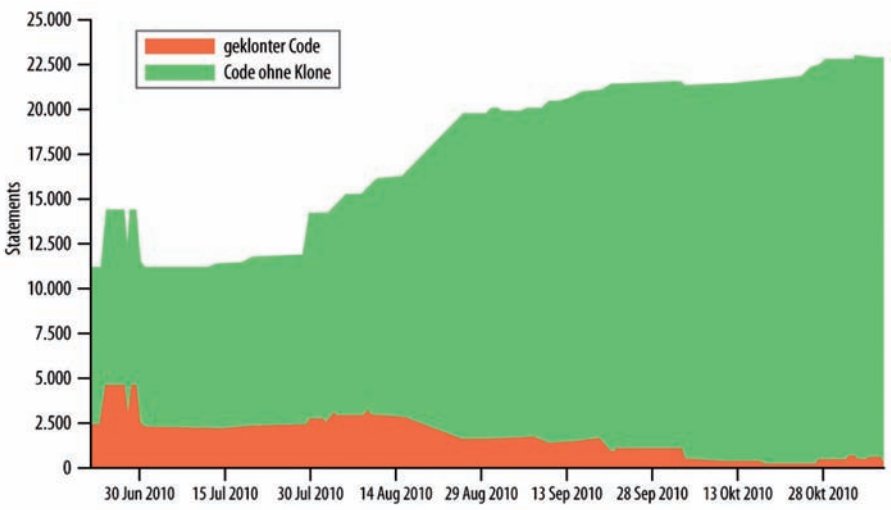

Abb. 5: Messbare Reduktion des Code-Clonings.

Best Practices

Im Rahmen unserer Tätigkeit konnten wir eine Vielzahl von Unternehmen in den verschiedenen Phasen der Einführung eines kontinuierlichen Qualitäts-Controllings unterstützen. Bei diesen Arbeiten haben sich einige Punkte als essenziell für den Erfolg des Controllings herausgestellt:

- *Frühzeitige Einbindung der Entwickler:* Kontinuierliches Qualitäts-Controlling lässt sich nur *mit* den Entwicklern und niemals gegen diese etablieren. Um Vorbehalte gegen zusätzliche Aufwände oder ein potenzielles Kontrollinstrument abzubauen, ist es wichtig, diese schon früh einzubinden. Dabei sollte transparent werden, welche Messwerte wie bestimmt werden, aber auch wie die Ergebnisse den einzelnen Entwickler unterstützen. Die meisten Entwickler sehen eine hohe Code-Qualität als wichtig an und sind oft sogar froh über jede Unterstützung bei der Verbesserung der Qualität. Insbesondere die langfristigen Trends in einem Dashboard können helfen, den positiven Einfluss von Verbesserungen über die Zeit zu verdeutlichen.
- *So viel Automatisierung wie möglich, so viel manuelle Inspektionen wie nötig:* Häufig stößt man bei der Einführung von Qualitäts-Controlling auf Vorbehalte, da anfangs nicht absehbar ist, dass sich die in die Qualität investierten Aufwände langfristig auszahlen. Wenn man diese initialen Aufwände durch einen hohen Automatisierungsgrad niedrig hält, kann das die Bedenken reduzieren. Gleichzeitig muss aber immer klar kommuniziert werden, dass eine vollständige Automatisierung nach dem aktuellen Stand von Forschung und Technik weder sinnvoll noch machbar ist (vgl. [Dei07]). Die Experteneinschätzung durch einen erfahrenen Entwickler lässt sich nicht ersetzen.
- *Flexible Werkzeuganpassung:* Jedes Projekt ist anders – und das spiegelt sich in den Details des Controlling-Prozesses und letztlich in der Konfiguration der eingesetzten Werkzeuge wider. Können die Werkzeuge nicht mit den durch das Projekt gegebenen Anforderungen mithalten, leidet darunter die Präzision und Relevanz der Messungen und letztlich der Erfolg des Qualitäts-Controllings. Beispiele für solche Anforderungen sind das Ausschließen von generiertem Code oder das Extrahieren von Abhängigkeiten aus den verschiedenen Artefakten (z. B. Plug-In-Konfigurationen) neben dem Code.

Auch wenn die Effekte größtenteils langfristiger Natur sind, lässt sich in den verschiedensten Projekten nach Einführung des kontinuierlichen Qualitäts-Controllings auch kurzfristig ein positiver Einfluss nachweisen. So haben wir ein gesteigertes Qualitätsbewusstsein bei den Entwicklern beobachtet, das nicht nur zur Verbesserung der Software führt, sondern auch zu konstruktiven Vorschlägen zur Optimierung des Controlling-Prozesses.

Noch klarer allerdings ist das Bild bei der Software selbst, bei der sich schon in relativ kurzen Zeiträumen zeigt, dass die Anzahl der Qualitätsdefekte nach Einführung des kontinuierlichen Qualitäts-Controllings messbar sinkt. **Abbildung 5** veranschaulicht dies am Beispiel der Entwicklung von Code-Cloning in einem betrieblichen Informationssystem in der Finanzdienstleistungsbranche. Obwohl das System durch die Implementierung neuer Funktionalität kontinuierlich wächst, reduziert sich der Anteil an Code, der Klone enthält (in rot dargestellt), auf ein vertretbares Minimum.

Unsere Erfahrung mit dem Qualitäts-Controlling mehrerer dutzend großer, langlebiger Systeme in unterschiedlichen Programmiersprachen zeigt, dass der beobachtete Effekt kein Einzelfall ist. Durch den kontinuierlichen, disziplinierten Einsatz von manuellen und automatischen Analyseverfahren lassen sich Qualitätsdefekte zu einem Zeitpunkt identifizieren, zu dem Korrekturen noch effizient durchführbar sind. Die Bereitschaft zur Qualitätsverbesserung vorausgesetzt, ermöglicht es das kontinuierliche Qualitäts-Controlling, den scheinbar unausweichlichen Qualitätsverfall effektiv zu verhindern. ■

Literatur

[Dei07] F. Deissenboeck, S. Wagner, M. Pizka, S. Teuchert, J.-F. Girard, An Activity-Based Quality Model for Maintainability, in: Proc. of the Int. Conf. on Software Maintenance 2007

[Dei08] F. Deissenböck, E. Jürgens, B. Hummel, S. Wagner, B. Mas y Parareda, M. Pizka, Tool support for continuous quality control, in: IEEE Software, 25(5):60–67, 2008

[ISO03] ISO. 9126-1 Software engineering – Product quality – Part 1: Quality model, International Standard, 2003

[Jür09] E. Jürgens, F. Deissenboeck, B. Hummel, S. Wagner, Do code clones matter? in: Proc. of the Int. Conf. on Software Engineering 2009

[Oma92] P. Oman, J. Hagemeister, Metrics for assessing a software system's maintainability, in: Proc. of the Int. Conference on Software Maintenance 1992

[Pau95] M. Paulk, C.V. Weber, B. Curtis, M.B. Chrissis, The Capability Maturity Model: Guidelines for Improving the Software Process, Addison-Wesley 1995

[Voa97] J. Voas, Can Clean Pipes Produce Dirty Water? in: IEEE Software, 14(4):93–95, 1997