Business Technology

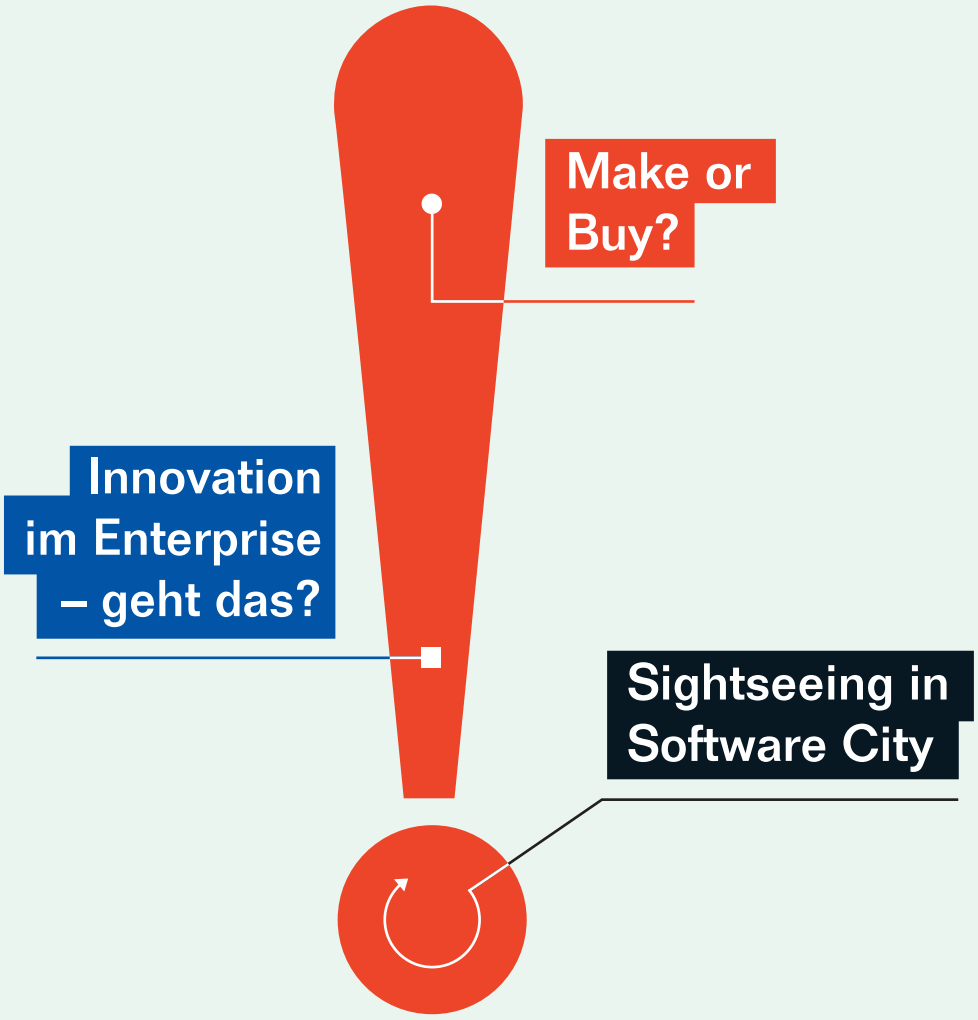
www.bt-magazin.de 3.2013 Heft 14

€ 4,90 S&S

Business Technology

Lean Innovation & Transformation

INNOVATIVE ARCHITEKTUR

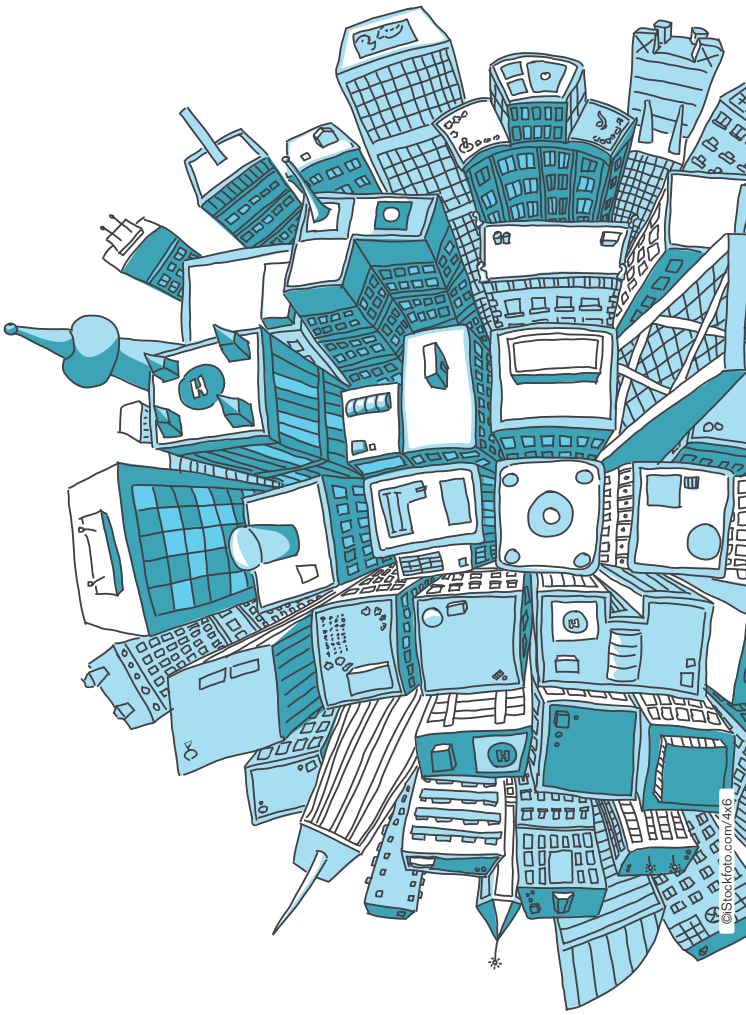


Softwarestädte erleichtern die Brennpunktsuche in komplexen IT-Systemen

Sightseeing in Software City

Softwarekarten geben gezielt Einblicke in Struktur, Beziehungen, Qualität und Entstehungsgeschichte von IT-Strukturen. In virtuellen Städten erkennen IT-Architekten Brennpunkte schneller, zuverlässiger und selbstständiger als bisher. Die Parallelen zu echten Stadtkarten, Flächennutzungsplänen und Bebauungsplänen überraschen selbst Softwareexperten: Die Metapher der Stadt eröffnet einen neuartigen Zugang zu den immer komplexeren Strukturen und Eigenschaften von Software.

AUTOREN: HUGH DOLLMAN UND MICHAEL HARRER



Die Herausforderungen bei der Softwareentwicklung steigen kontinuierlich: Softwarearchitekturen großer Unternehmen werden immer aufwändiger zu bewerten, zu steuern und zu pflegen. Gleichzeitig stellen sie erhebliche Investitionen dar, und ihr Beitrag zum Unternehmenserfolg nimmt zu. Um dem steigenden Rationalisierungsdruck und immer höheren Qualitätsanforderungen gerecht zu werden, müssen erfolgskritische Informationen in kürzester Zeit aufbereitet und ausgewertet werden können.

Das Projekt „Softwarelandkarten für Systemaudits“ des Münchner IT-Beratungs- und Softwareentwicklungsunternehmens MaibornWolff et al in Zusammenarbeit mit der Brandenburgischen Technischen Universität Cottbus sorgt mit der Stadtmetapher für einen neuartigen Zugang und schafft einen ganzheitlichen Überblick über die Entwicklung von Softwaresystemen. Die Stadtkarten führen wesentliche Daten, wie zum Beispiel Strukturinformationen, Komplexitätsmetriken, Abhängigkeiten und die Bauaktivitäten der Entwickler zusammen.

SOFTWAREVISUALISIERUNG UNTERSTÜTZT SYSTEMAUDITS

In Systemaudits bewerten IT-Architekten ein Softwaresystem. Ihr Ziel ist es, klare Handlungsempfehlungen abzuleiten. Dazu müssen sie das System zunächst verstehen und dann analysieren. Softwarelandkarten visualisieren die statischen Strukturen und die Beziehungen innerhalb des Systems. Bei dieser Analyse ist es extrem hilfreich, wenn Qualitätskennzahlen und Bauaktivität auf einen Blick erfasst werden können. Denn solche „Hotspots" bringen Fehlerrisiken mit sich – es lohnt sich, sie im Auge zu behalten. Wichtig ist es auch, zentrale und große Module zu identifizieren und die Abhängigkeiten zwischen den Modulen zu erfassen.

Moderne IT-Systeme sind mit herkömmlichen Methoden nicht mehr überblickbar. Probleme werden nicht frühzeitig erkannt. Die Ursache liegt in der zunehmenden Größe und Komplexität von Softwaresystemen. Herkömmliche Methoden, Systeme und Qualitätsmerkmale zu visualisieren, scheitern an ihren natürlichen Grenzen:

- UML-Darstellungen sind feingranular und werden sehr schnell unübersichtlich. Sie sind oft zu detailliert und veraltet.
- Reports geben Auskunft über spezifische Fragen. Sie vermitteln keinen Überblick über ein System und helfen nicht, das System in seiner Ganzheit zu verstehen.

Die Visualisierung von Software ist der Königsweg, wenn unterschiedlichste Detailtiefen erfasst werden sollen. Das Bild der Softwarekarten bietet sich an, denn alle Menschen kennen Karten und wissen spontan, wie sie funktionieren. Seit Google Earth ist es selbstverständlich, dass man Ausschnitte aus großen Karten überfliegt und hineinzoomt, um Details zu erkennen. Auf Anhieb lassen sich Anomalien in den Karten erfassen, also Bereiche, die sich deutlich vom Rest unterscheiden.

VISUALISIERUNGEN MÜSSEN HOHEN ANSPRÜCHEN GENÜGEN

Visualisierungen von Software sollten für möglichst unterschiedliche Betrachtungen nutzbar sein. Dazu müssen sie anspruchsvolle Anforderungen erfüllen:

- Jedes abgebildete Softwaresystem sollte so dargestellt werden, dass der Betrachter sich schnell darin zurechtfindet und die spezifischen Charakteristika des Systems klar aus der Visualisierung hervorgehen.
- Die Visualisierung sollte verschiedene Detaillierungsgrade zulassen und zu unterschiedlichen Entwicklungszeitpunkten ebenso Architekturelemente wie auch Codekomponenten auf vergleichbare und einheitliche Weise sichtbar machen.
- Gegenüber kleineren Änderungen des Softwaresystems sollte sich die Visualisierung als hinreichend stabil erweisen – schließlich muss sie die Beobachtung sich entwickelnder Systeme über verschiedene Versionen unterstützen, ohne dass das Gesamtbild beeinträchtigt wird.
- Verschiedene Datenquellen müssen konsistent und systematisch eingebunden werden können.
- Der Visualisierungsvorgang und die Visualisierungen selbst sollten von kleinen auf große Systeme mit vielen Millionen Codezeilen skalierbar sein.

Visualisierungen sollten das strukturelle Wachstum eines Systems verfolgen können. IT-Architekten profitieren dann zum Beispiel davon, einmal wöchentlich ihr System zu plotten und aufzuhängen. So sehen alle Teammitglieder, was sie zusammen erreicht haben. Ideal ist es, wenn auf eine sehr ansprechende Weise navigiert werden kann und Beziehungen sichtbar werden: Im Überflug erkennen Softwareentwickler gefährliche Zyklen. Neue Mitarbeiter erhalten schnell einen Überblick über große Projekte und arbeiten sich zügig ein. Eine visuelle Darstellung fungiert auch als Risiko- und Aufwandsindikator durch die Darstellung von Codehistorien. Schnell wird erkannt, wie viele Änderungen eine Klasse durchlaufen hat und wie viele Bearbeiter daran mitgewirkt haben. Das Know-how in verschiedenen Entwicklerteams wird durch die Darstellung von Autorenschaften transparent. Gezielt lassen sich typische Problemstellungen analysieren: Welche Teile des Systems sind betroffen, wenn Bibliothek X entfernt oder Modul Y umgebaut wird?

STADTBILDER SIND UNS VERTRAUT

In der Stadt als Metapher für Planung und Entwicklung erkennen Softwareentwickler und IT-Architekten sofort die entscheidenden Parallelen. Wie eine Stadt unterliegt eine Softwarearchitektur bestimmten Entwicklungsphasen. Sowohl in der realen wie der virtuellen Stadt sprechen Experten von Entwurf, Konstruktion, Bau, Erhaltung, Weiterentwicklung und Sanierung. In beiden Welten müssen die Spezialisten Entwicklungsprozesse und Artefaktstrukturen verstehen. Verortungs- und Orientierungssysteme sind das räumliche Gedächtnis für Architekten der realen Welt wie auch für Softwareentwickler. Beide Expertengruppen nutzen Darstellungs- und Abstraktionstechniken und eine möglichst kohärente Kommunikation. Beide suchen Orientierung in den Strukturen.

Städte und größere urbane Strukturen sind uns intuitiv vertraut. Mithilfe der zahlreichen Analogien geben

sie Softwaresystemen und Entwicklungsgeschichten eine Gestalt. Sobald eine Stadtkarte die Softwarearchitektur veranschaulicht, werden die Slums erkannt. Durch Reinzoomen werden einzelne Stadtteile und Gebäude analysiert. Der Betrachter gleitet im Tiefflug durch die urban dargestellte Infrastruktur des Softwaresystems. Ein gehöriger Spaßfaktor ist dieser sehr intuitiven Bewegung nicht abzusprechen; in den Testläufen haben sich die Projektinitiatoren von MaibornWolff et al an leuchtende Gesichter gewöhnt.

Die Parameter, nach denen Farbe, Breite, Höhe und Positionierung der Gebäude festgelegt sind, können je nach Bedarf variieren. Damit ist die Visualisierung flexibel und kann je nach Zielgruppe angepasst und auf die unterschiedlichsten Aspekte hin ausgelegt werden. Es ist wichtig, die Konfigurationen stabil zu halten, damit man sich leicht orientiert. Dann erkennen auch Nichtfachleute, wo die Stadt gesund wächst und wo Handlungsbedarf besteht. Kaufmännische Verantwortliche und Vorstände bis hin zu Investoren erhalten einen visuellen Eindruck von der ansonsten unsichtbaren Architektur.

GROSSE ABBILDUNGSBREITE UND -TIEFE IST ESSENZIELL

Die Idee an sich hat eine lange Geschichte. Doch erst jetzt sind die Softwarearchitekturen großer Unternehmen so heterogen und komplex geworden, dass man ihrer auf konventionelle Weise, also mit schriftlichen Reports oder langen Zahlenkolonnen, nicht mehr Herr wird.

Von anderen Lösungen unterscheidet sich die von hier beschriebene vorangetriebene Entwicklung durch ihre Abbildungsbreite und -tiefe. Während heute auf dem Markt erhältliche Systeme lediglich Strukturen zeigen und Daten, die diesen Strukturen direkt zugeordnet werden können, vermittelt die neue Lösung durch Analyse und Visualisierung der Beziehungen die echte Komplexität. Sie visualisiert die komplette Systemwelt – also ganze Städte statt nur Stadtteile, um im Bild zu bleiben.

Die Komplexität wird vollständig abgebildet, sehr ausdrucksstark und in Echtzeit, aber ohne Interpretation. Die Interpretation selbst basiert nicht nur auf Kennzahlen, sondern auf der Verbindung von Kennzahlen und deren Abhängigkeiten. Zur besseren Risikobewertung werden Größen herangezogen, wie Bauaktivität,

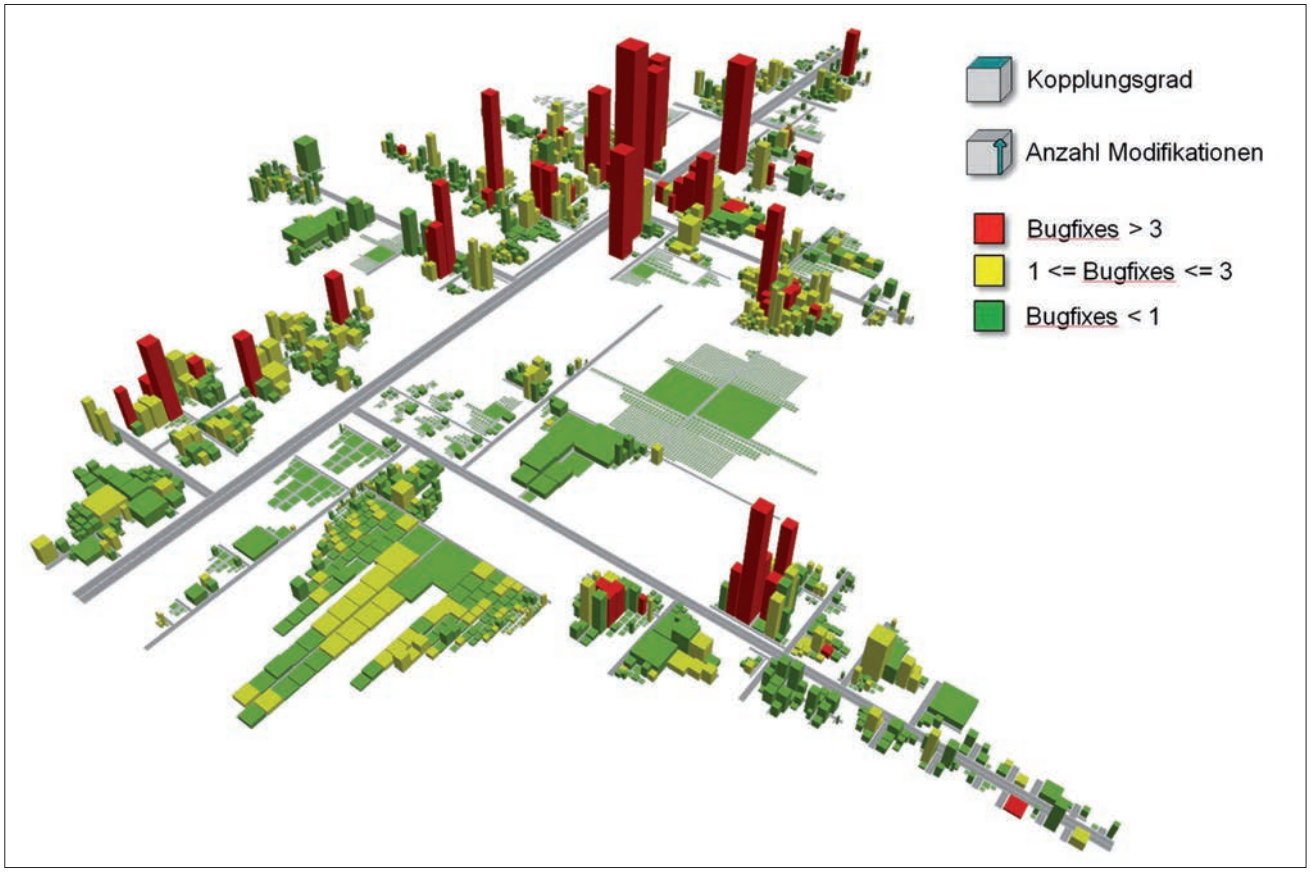


Abb. 1: Bauaktivitäten zeigen kritische Systemteile auf

lokale Codequalität, Testabdeckung und Anzahl der Autoren. Die Interpretation hängt auch vom Architekturverständnis selbst ab. Hier spielen Zyklen, Modulgrößen, Modulschnitte und Abhängigkeiten zwischen Softwareschichten eine Rolle.

Beim Systemaudit erkennen Betrachter der unterschiedlichsten Fachrichtungen potenzielle Brennpunkte unabhängig, unparteiisch und schnell. Die Karten unterstützen die Kommunikation zwischen Auditoren, Projektverantwortlichen, Architekten und Entwicklern. Gemeinsam korrelieren sie Bauaktivität mit Komplexität. Kritische Systemteile fallen sofort ins Auge, sodass von erfahrenen Architekten Handlungsbedarf abgeleitet werden kann (**Abb. 1**).

BESONDERS NÜTZLICH SIND KARTOGRAFIERTE BEZIEHUNGEN

Das Visualisierungstool erstellt Beziehungskarten: Die Nähe von städtischen Strukturen zu anderen Städten entspricht dem Kopplungsgrad. Aus- und eingehende Abhängigkeiten zu anderen Modulen sind unmittelbar zu sehen, auf Wunsch visualisiert durch Fahrzeugverkehr. Dadurch entsteht ein Gesamtüberblick der Größen von Systemmodulen. Systembeziehungen wie Zyklen und Abhängigkeiten werden schnell und intuitiv exploriert. Der Erklärungsbedarf fällt minimal aus.

Informationen, die nicht automatisch gesammelt werden, ergänzen die Auditoren nachträglich. Das können zum Beispiel Hinweise und Beobachtungen sein, die das Interpretieren erleichtern und die Darstellung um wichtige Aspekte anreichern. Solche Annotationen eignen sich beispielsweise zum Herausheben und Dokumentieren wichtiger Brennpunkte.

Der geplante stufenlose Zoom wird wie bei Google Maps einen Kontrollflug vom Gebäudekomplex bis in die kleinsten Bauteile erlauben, also vom System über Komponenten, Pakete und Klassen bis hin zum Code. Auf diese Weise wird der fachliche Schnitt der Module klarer und der Bezug zur Fachlichkeit wird verstärkt. Architekturverletzungen sind auf Anhieb sichtbar. Zusätzliche Informationen, wie zum Beispiel die System- und Modulkritikalität, Stakeholder und Architekturtreiber oder Entwicklungsprozess lassen sich für ein Audit zusätzlich einarbeiten. Diese Eigenschaften sind allerdings in der Karte nur schwer abbildbar, da sie den Softwarekomponenten nicht einfach zugeordnet werden können.

Der Kartenersteller bestimmt die Granularität der Information selbst. Die Parameter und ihre Darstellung lassen sich mit wenigen Klicks anpassen. Thematische Karten zeigen bei vorhandenen Grunddaten beispielsweise Testabdeckung, Fehlerdaten, Fehlerbehebung, Profiling-Daten, Codemetriken, wie Kommentardichte und Kopplungsgrade oder Revisionsstände. Multidimensionale Produkt- und Prozessdaten lassen sich in einer einheitlichen Basisstruktur kombinieren.

Die Softwarelandkarten nutzen das gesamte kartographische Darstellungsrepertoire, also die intuitiv beherrschbare Semiotik der Landkarten. Begrifflichkeiten und Darstellungstechniken für urbane Strukturen werden systematisch auf die Konstruktion und das Entwicklungsmanagement von Software übertragen:

- Einbettung gewohnter Konzepte der Softwarearchitektur in die Stadtmetapher
- Identifikation und Darstellung der Ordnungs- und Regelsysteme für Softwaresysteme, die eine urbane Struktur und Entwicklung prägen
- Architekturen als Ergebnis historischer Prozesse

DER AUFBAU DER KARTEN IST FLEXIBEL

Wie Karten von Ländern und Städten sind Softwarelandkarten im Aufbau flexibel. Beim einfachsten Modell entspricht ein Gebäude einer Klasse. Klassen, die an der gleichen Straße liegen, gehören zum gleichen Package. Bei der Definition einer bestimmten Package-Tiefe werden Gebäude zu einem Modul zusammengefasst. Eine beispielhafte Package-Struktur könnte etwa so aussehen:

- *de.mwea.systemA.subsystemX.services*
- *de.mwea.systemA.subsystemX.services.transferobjects*
- *de.mwea.systemA.subsystemX.manager*
- *de.mwea.systemA.subsystemX.dao*
- *de.mwea.systemA.subsystemX.utils*

Eine derart einfach aufgebaute Karte wird abhängig von der Struktur eines Systems gezeichnet. Sie gibt keinen Aufschluss über die Beziehungen, die zwischen verschiedenen Subsystemen existieren – zum Beispiel, welche Vororte am meisten miteinander zu tun haben (**Abb. 2**).

Aufwändigere Karten können auch so gezeichnet werden, dass die örtliche Nähe einzelner Stadtteile das Maß für die Kopplung der Module darstellt. Dazu werden so genannte *energiebasierte Layouts* verwendet, die von der BTU Cottbus entwickelt wurden (**Abb. 3**).

Bewährt hat sich die Darstellung von Klassen als Gebäude. Die Gebäudegrundfläche entspricht der Bedeutung der Klasse im System. Sie gibt das Maß für die

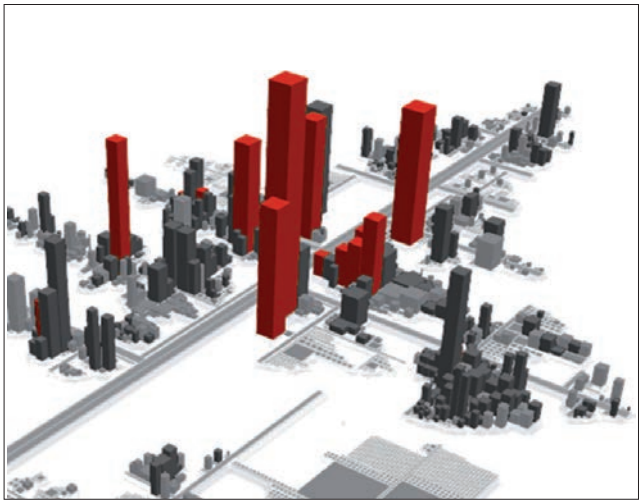


Abb. 2: Beziehungskarten machen Kopplung beherrschbar

Abhängigkeit vom Restsystem wieder. Der Fachbegriff für diese Kennzahl ist die Kopplung, die Summe der eingehenden und ausgehenden Abhängigkeiten.

Eine Klasse beispielsweise, die keine andere Klasse braucht und auch von keiner anderen Klasse benötigt wird, hat eine sehr kleine Grundfläche. Eine Klasse, die 10 000 andere Klassen braucht, aber von keiner anderen Klasse benötigt wird, verfügt über die gleiche Grundfläche wie eine Klasse, die von 10 000 anderen Klassen benötigt wird, selbst aber keine andere Klasse benötigt. Die Höhe der Gebäude entspricht beispielsweise der Größe der Klasse, gemessen an der Anzahl an Zeilen oder Methoden. Alternativ kann man etwa der Gebäudehöhe eine Komplexitätskennzahl zuweisen, wie zum Beispiel die zyklomatische Komplexität.

Farben können als Warnzeichen genutzt werden. Farbänderungen hängen dann von einem vordefinier-

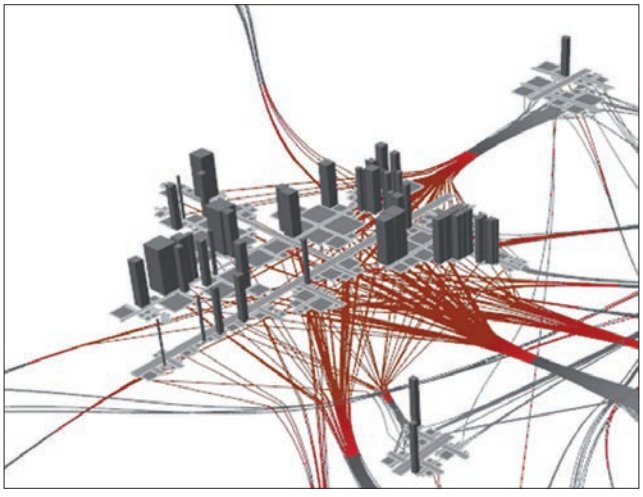


Abb. 3: Strukturkarten identifizieren Brennpunkte

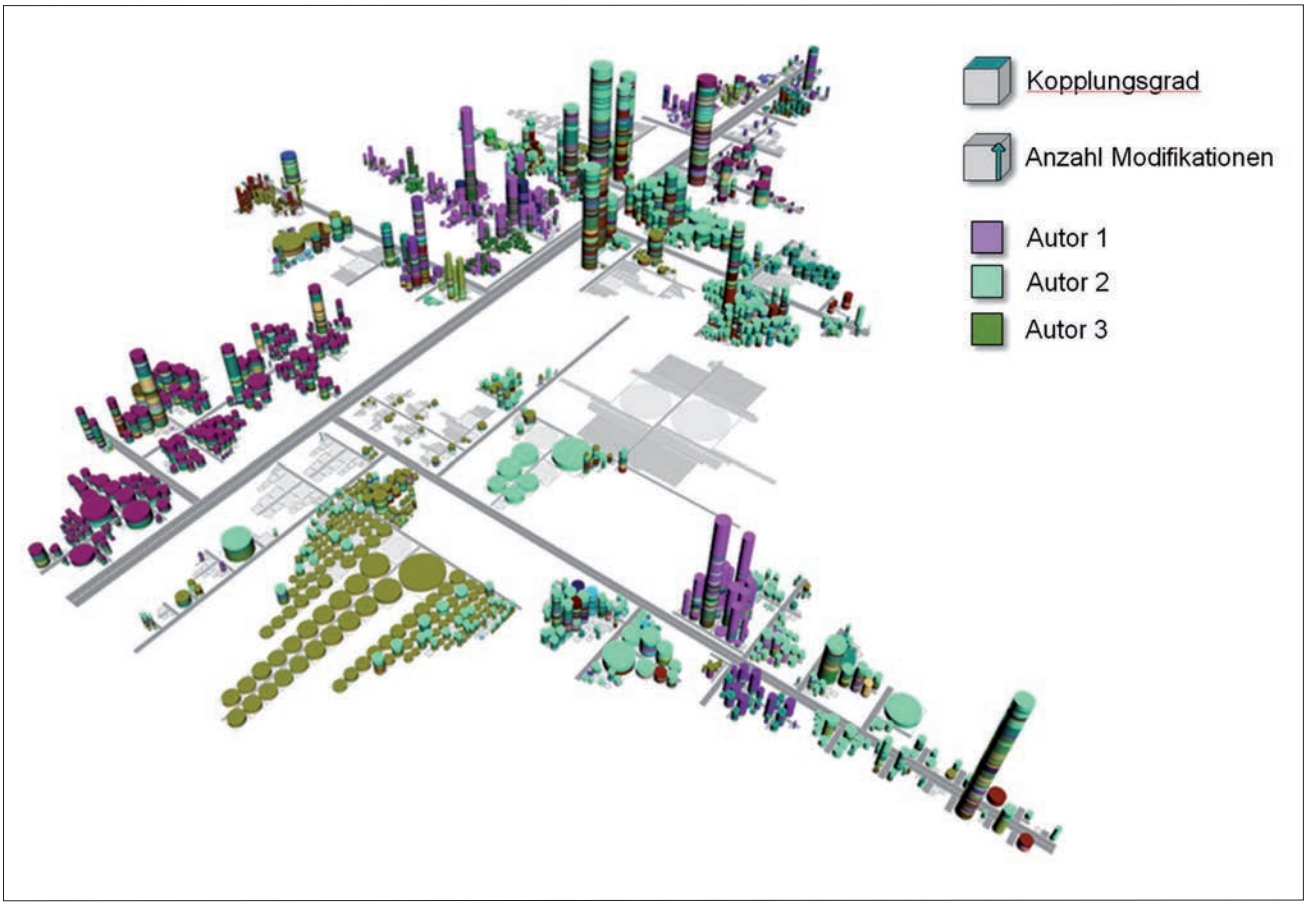


Abb. 4: Strukturkarten identifizieren Brennpunkte

ten Schwellwert ab. Farbe kann auch den vordefinierten Schwellenwert einer anderen Kennzahl wiedergeben. Möchte der Systemauditor beispielsweise Klassen finden, die groß sind und häufig verändert wurden, kann er die Änderungshäufigkeit mithilfe der Farbe abbilden – er würde dann gezielt nach hohen, roten Gebäuden suchen.

ZAHLREICHE ARTEFAKTE EIGNEN SICH FÜR EINE SOFTWARESTADT

Softwarestädte leben vom Fokus auf relevante Auditdaten und deren Integrierbarkeit. Jede Kennzahl muss numerisch sein und einem Softwareartefakt zugeordnet werden. Je mehr Artefakte zur Analyse bereitstehen desto aussagekräftiger werden die Karten. Ein Softwarestand als Bytecode, also zum Beispiel ein Release, ist die Voraussetzung zur Erstellung einer initialen Landkarte. Sie dient der Darstellung von Abhängigkeiten zwischen Klassen und Paketen und der Visualisierung der Codestruktur. Die Abbildung der zeitlichen Entwicklung ist bei mehreren Projektständen möglich. SVN-Logs ermöglichen die Zuordnung der Autoren zum Code und die Visualisierung von Wissensverteilung. Sie helfen außerdem bei der Verknüpfung fachlicher Änderungen zu den als Stadtteile abgebildeten Modulen. Übliche Änderungen sind User Stories, Defects und Anforderungen.

Durch Zuordnungen von Revisionen zu Releases lassen sich zeitlich interessante Landkarten vor Code-Freezes und in Hotfix-Phasen erstellen. Darüber hinaus können neue Features und Trends abgebildet werden. Projektspezifische Analyseartefakte, beispielsweise von PMD, Checkstyle, Findbugs oder SonarJ, können als Datenbasis für Landkarten verwendet werden. Weitere Darstellungen entstehen durch die Visualisierung von Performance-Logs, die Verknüpfung der Landkarten mit Bugtracking-Tools wie zum Beispiel Jira, die Abbildung von Testabdeckung und die Verknüpfung weiterer Kennzahlen mit dem Code. Je mehr Artefakte zur Verfügung stehen, desto aussagekräftiger werden die Karten – grundsätzlich ist alles abbildbar.

ZUM TEIL NOCH ZU FUSS VON DER SOFTWARE ZUR STADT

Die Basis der Softwarestädte bilden im Augenblick noch ausschließlich kompilierbare Java-Projekte. Beim Im-

port von Softwareartefakten in das Analysewerkzeug wird ein Modell der Software aufgebaut. Dieses Modell wird mit zusätzlichen Daten sukzessive angereichert. Bislang ist die Erstellung dieses Modells noch nicht automatisiert. Der Kartenersteller muss vorher definieren, welche Packages betrachtet werden sollen. Die Verknüpfung der Artefakte, also etwa die Zuordnung eines kompilierten Releases zu einer SVN-Revision, ist auch noch Handarbeit. Ins System ist ein Dependency Finder eingebunden. Er arbeitet mit Java-Class-Dateien und extrahiert Hierarchien von Systemen, Paketen, Klassen und Methoden, die Abhängigkeiten zwischen Komponenten und Basiskennzahlen, wie beispielsweise die Anzahl der Methoden und Attribute. Eingangsformate sind **.jar*, **.class*, **.ear*, **.war* usw. Das Ergebnis wird als XML-Datei exportiert.

Nach der Verknüpfung mit dem SVN-Log wird interpretierbar, welche Modifikationen es an einer Komponente gab, welche Autoren sie manipuliert haben und welche Änderungskopplungen im Verlauf der Manipulation entstanden sind. Die statische Codequalität kann mit externen Tools erfasst und mit dem Model verknüpft werden. Noch werden die Daten manuell zusammengeführt. Dieser Prozess wird in Zukunft in den Build-Prozess integrierbar sein.

DIE SOFTWARESTADT ENTSTEHT IN DREI SCHRITTEN

Softwarestädte werden in einem dreistufigen Prozess generiert. Jede Stufe arbeitet mit einem spezifischen Modell. Das Primärmodell erfasst die Struktur eines Softwaresystems, seine Datenflüsse und seine Entwicklung über einen definierten Zeitraum. Primärmodelle sind die logischen Basismodelle aller Softwarevisualisierungen.

Das Sekundärmodell, wie zum Beispiel das neue Evo Streets, ergänzt geometrische Informationen auf der Basis hierarchischer Systemdekompositionen und des Komponentenalters. Es liefert geometrisch stabile Straßenkarten für wachsende Softwaresysteme und stellt die Entwicklungshistorie in Form von unterschiedlichen Höhenniveaus dar. Diese Darstellung ist intuitiv erfassbar, hat aber auch Nachteile: Systeme, die häufigem Wandel unterliegen oder viele Subsysteme beherbergen, werden eher weitläufig als kompakt dargestellt. Für die Anordnung der Gebäude in der Stadt und die Platzierung neuer Gebäude gibt es unterschiedliche Algorithmen. Ein kompaktes Layout findet in der Regel den meisten Zuspruch, sodass eine Stadt nicht in die Länge wächst und die Karte gleichmäßig ausgefüllt wird.

Tertiärmodelle leiten sich mithilfe von Transformationen aus Sekundärmodellen ab. Das sind zum Beispiel Projektionen, Farbgebung oder übergelegte Symbole und Diagramme. Solche thematischen Softwarekarten dienen in der Regel spezifischen Anwendungsszenarien. Um möglichst vielfältige Moduleigenschaften in einheitlicher Form darzustellen, hat das Projektteam so genannte *Eigenschaftstürme* entwickelt, die derzeit mit verschiedenen Partnern aus der Industrie diskutiert werden.

DEN NACHTEILEN STEHT GROSSES POTENZIAL GEGENÜBER

Um Karten anreichern zu können, müssen die Informationen in ausreichender Qualität vorhanden sein. Viele Unternehmen sind allerdings nicht sehr diszipliniert bei der Archivierung ihrer Build-Artefakte oder der Zuordnung von SVN-Ständen zu Build-Artefakten. Umzüge, große technische Veränderungen oder die Veränderung vom Codeeigentum können auch dazu führen, dass Teile der Historie des Codes gar nicht mehr verfügbar sind. Die Aussagen werden dadurch verzerrt. Teilweise bestehen Bedenken bei der Herausgabe von Binaries und Sourcecode. Außerdem schaffen Softwarelandschaften eine völlig neue Transparenz. Das wird nicht überall auf Zustimmung stoßen. Würden kritische Infrastrukturteile der Softwarearchitektur beispielsweise Einzelpersonen zugeordnet werden, greift der Datenschutz. Bei angemessenem Einsatz überwiegen die Vorteile. Das Potenzial ist enorm und die Anwendungsgebiete wachsen stetig (**Abb. 4**).

Hugh Dollman

ist Senior Software Engineer bei der MaibornWolff et al GmbH in München. Sein Wirkungsbereich liegt im Umfeld von Automotive, Kundenbeziehungs- und Bonusprogrammen.

Michael Harrer

arbeitet seit 2010 als Senior Software Engineer bei MaibornWolff. Er beschäftigt sich vorrangig mit der Implementierung von Softwaresystemen mit objektorientierten Programmiersprachen.