CrossMark

# *Empirische Forschung zu Software-Evolution*

*Bestandsaufnahme und Vision aus dem DFG-Schwerpunktprogramm*
**Design for Future – Managed Software Evolution**

**Barbara Paech · Sven Apel**
**Lars Grunske · Christian Prehofer**

## Einleitung

Engels et al. [5] argumentieren, dass bei der Erstellung von Software zu kurzfristig gedacht wird. Sie fordern unter dem Motto „Design for Future" ein Umdenken bei EntwicklerInnen, ProjektleiterInnen und dem Management, sodass die gesamte Lebenszeit der Software und damit auch die Änderbarkeit und Wartbarkeit der Software berücksichtigt werden sollten. Weiterhin fordern sie gezielte Forschungstätigkeit, insbesondere „Metamodelle für die Integration von Wissen über Software im Bereich Softwarearchitektur, aber ebenso in anderen Bereichen wie etwa für das Testmanagement" sowie „Grundlagenforschung, Methoden, Metriken, Werkzeuge und Prozesse, die die Koevolution von Systemen mit ihren sich verändernden Anforderungen, Randbedingungen und Plattformen ermöglichen" und empirische Untersuchungen zur Validierung der Ergebnisse.

Mit diesem Ziel forschen deutsche Softwaretechnik-ExpertInnen seit 2012 in 13 Projekten des DFG-Schwerpunktprogramms *Design for Future – Managed Software Evolution (SPP 1593)* [http://www.dfg-spp1593.de]. Die AutorInnen des vorliegenden Beitrags gehören zu den ProjektleiterInnen. Das Anliegen dieses Beitrags ist, aus den Diskussionen innerhalb des SPP 1593 die Wichtigkeit und die Herausforderungen empirischer Forschung zu Software-Evolution zu begründen und eine Vision für zukünftige Forschung in diesem Bereich vorzustellen.

Wir beginnen diesen Beitrag mit zwei Beispielen für das Dilemma zwischen kurzfristigem und langfristigem Erfolg von Software. Aufbauend auf einigen grundlegenden Begriffen zu empirischer Forschung und Software-Evolution arbeiten wir die Ziele und Herausforderungen von empirischer Forschung zu Software-Evolution heraus. Dann berichten wir von einer Umfrage zu diesen Themen innerhalb des SPP 1593. Wir schließen mit unserer Vision von empirischer Software-Evolutionsforschung und der daraus abgeleiteten Forderung für eine verstärkte Förderung.

**Kurzfristiger und langfristiger Erfolg von Software.** Software muss in ihrem Lebenszyklus kontinuierlich weiterentwickelt werden. Dies ist notwendig, da sich das Umfeld der Software, also die Hardware, der Einsatzbereich und damit die Anforderungen an die Software kontinuierlich weiterentwickeln. Software wird immer wieder in neuen, oft nicht geplanten Szenarien eingesetzt, die z. B. Anpassungen der ursprünglichen Architektur erfordern. Dies führt allerdings zu einem typischen Dilemma: Es ist für den kommerziellen Erfolg einer Software wichtig, dass diese möglichst schnell auf den Markt kommt (vgl. Interview Jan Bosch, http://leanmagazine.net/lean/jan-bosch-the-need-for-speed/). Dies führt

DOI 10.1007/s00287-015-0910-0


Barbara Paech
Universität Heidelberg, Heidelberg
E-Mail: paech@informatik.uni-heidelberg.de

Sven Apel
Universität Passau, Passau
E-Mail: apel@uni-passau.de

Lars Grunske
Universität Stuttgart, Stuttgart
E-Mail: lars.grunske@informatik.uni-stuttgart.de

Christian Prehofer
fortiss GmbH & TU München, München
E-Mail: prehofer@fortiss.org

## Zusammenfassung

Software-Evolutionsforschung ist essenziell, um langlebige Software verlässlich und effizient zu entwickeln. Dieser Beitrag plädiert auf Basis der Erfahrungen aus dem DFG-Schwerpunktprogramm (SPP) „Design for Future – Managed Software Evolution“ für eine stärkere Rolle von empirischer Forschung zu Phänomenen im Kontext von Software-Evolution und der Wirksamkeit von neuentwickelten Methoden und Werkzeugen in diesem Bereich. Dazu werden die Ziele und Herausforderungen dieser Forschungsrichtung vorgestellt und die Ergebnisse einer Umfrage im SPP präsentiert. Darauf aufbauend wird eine Forschungsvision entwickelt, welche Post-mortem-, In-vitro- und In-vivo-Studien nutzt, um sicherzustellen, dass neue Ansätze zur Unterstützung von Software-Evolution umfassend evaluiert werden und eine nachhaltige Umsetzung in die Praxis ermöglicht wird.


zwangsläufig zu Kompromissen, oft in der Qualität, aber auch in der Vorausplanung für künftige Erweiterungen. Natürlich ist Qualität für Software ein Erfolgskriterium. Vorausplanung für die Zukunft, also die Unterstützung von Evolution, ist jedoch für den kurzfristigen Erfolg nicht notwendig. Ohne die Berücksichtigung von Evolution ist aber der langfristige Erfolg stark gefährdet.

Es gibt einige bekannte Beispiele für diese Entwicklung, auch wenn systematische wissenschaftliche Untersuchungen fehlen. Ein Beispiel ist der erste Internet-Browser, Netscape, der aus dem Protoyp Mosaic der University of Illinois hervorgegangen ist. Dieser wurde 1994 auf den Markt gebracht, konnte nach einigen Jahren der (selbst initiierten) Innovationsgeschwindigkeit jedoch nicht mehr folgen und die Weiterentwicklung wurde eingestellt. Laut Jamie Zawinski, einem der tragenden Entwickler, waren die Aufwendungen für die Evolution zu hoch: „It constituted an almost-total rewrite of the browser, throwing us back six to ten months“ (http://www.jwz.org/gruntle/nomo.html).

Eine weitere Beobachtung ist, dass komplexe Softwaresysteme oft ein wesentliches Wirtschaftsgut von Firmen verkörpern und die Evolution dieser für den Erfolg von Firmen essenziell ist. Ein Beispiel hierfür ist Nokia. Nokia hatte Ende der 90er-Jahre die Softwareplattform Symbian S60 für Smartphones etabliert und hatte damit im Höhepunkt der Entwicklung, 2008, fast 40 % Marktanteil. Symbian war als flexible Softwareproduktlinie aufgebaut und ermöglichte die schnelle Anpassung auf verschiedene Produkte in vorher nicht gekannter Zeit. Als dann jedoch der Markt durch Konkurrenten ab 2007 auf einfache Nutzerschnittstellen mit Touchscreens (ohne Bedienstift) umschwenkte, konnte Nokias komplexe Software nicht rechtzeitig angepasst werden. Trotz eines enormen Aufwandes von geschätzt 6000 EntwicklerInnen war es nicht möglich, diesem Evolutionsdruck zu folgen und die Plattform wurde vor wenigen Jahren aufgegeben (http://www.asymco.com/2011/02/04/nokia-employs-as-many-engineers-for-symbian-and-meego-as-apple-does-for-all-its-product-lines/).

Diese Beispiele zeigen, dass es nicht möglich ist, die Entwicklungen der Zukunft vorherzusehen, aber auch, dass Software auf Evolution hin ausgelegt sein muss. Wie von Engels et al. [5] dargestellt, gibt es unterschiedliche Ansätze für Methoden und Werkzeuge, die die Evolution von Software unterstützen. Diese werden im SPP 1593 in mehreren Teilprojekten untersucht und weiterentwickelt. Es fehlt aber ein umfassendes Verständnis der Phänomene, die im Kontext von Software-Evolution auftreten, speziell im Hinblick auf die Anwendung in großen Softwaresystemen, die über längere Zeit entwickelt und betrieben werden. Dieses Verständnis ist nur durch einen signifikanten Ausbau der empirischen Forschung in diesem Bereich zu gewinnen, um damit auch die Güte der entwickelten Modelle, Methoden und Werkzeuge nachweisen zu können. Dies ist das wesentliche Thema dieses Artikels und wird im Folgenden aus der Literatur und den Erfahrungen des SPP 1593 heraus beleuchtet.

## Software-Evolution und empirische Forschung

**Der Begriff Software-Evolution** wird in der Literatur nicht einheitlich verwendet. Wie von Mens et al. [10] beschrieben, wird Software-Evolution oft als Synonym von Software-Wartung angesehen, d. h. Software-Evolution umfasst alle Aktivitäten zur Behebung von Softwarefehlern und zur Weiterentwickelung von Software. Teilweise wird aber der Begriff Software-Evolution nur für die Weiterentwicklungs-

**Abstract**
Research on software evolution is essential for the development of reliable and efficient, long-living software. In this article, we argue based on the experiences in the DFG-priority programme (SPP) "Design for Future – Managed Software Evolution" for empirical research on phenomena of software evolution and on the merits of new methods and tools in this area. Therefore, we discuss aims and challenges of this field of research and report of an online survey among members of the SPP. We then propose our vision of post-mortem, in-vitro, and in-vivo studies to ensure a comprehensive evaluation of new approaches to software evolution and their sustainable application in practice.

aktivitäten verwendet. Wir folgen hier der Definition von Godfrey et al. [6]:

– Software-Evolution umfasst alle Phänomene, die mit der Veränderung von Software zu tun haben.

Im Gegensatz zur Software-Wartung betont Software-Evolution die Entwicklung neuer Entwürfe, ungeplante Phänomene und wissenschaftliche Fragestellungen zu Charakteristiken von Softwareänderungen. Godfrey et al. [6] und andere fordern dementsprechend, dass Forschung zu Software-Evolution Modelle hervorbringt, die die Vergangenheit, Gegenwart und Zukunft einer Software und ihrer Änderungen beschreiben können. Ein wissenschaftlicher Zugang zu solchen Modellen umfasst die Beobachtung der Phänomene, die Erstellung von Modellen und Hypothesen, sowie das Testen der Hypothesen und die Anpassung der Modelle. Genau dies geschieht durch empirische Forschung im Software Engineering.

**Empirische Forschung im Software Engineering.** Empirische Forschung wird typischerweise als Domäne der Sozial- und Lebenswissenschaften angesehen und ist in der Informatik nicht stark etabliert. So gibt es z. B. wenige Studiengänge der Informatik, die empirische Methoden explizit vermitteln. Im Bereich Software Engineering wird dieses Thema seit den 90er-Jahren verstärkt diskutiert, international angetrieben durch Vic Basili (z. B. [2]), im deutschen Raum insbesondere angetrieben durch Dieter Rombach und Walter Tichy (z. B. [4, 13]). Dabei kommen vor allem zwei Forschungsmethoden zur Anwendung:

– *Experimente*, um in einer kontrollierten Umgebung spezifische Hypothesen und Effekte zu untersuchen [14],
– *Fallstudien*, um in einem komplexen Umfeld gezielt Phänomene zu beobachten und zu analysieren [12].

Im Rahmen von *Design Science* [7] wird Software-Engineering-Forschung, die Methoden und Werkzeuge zur Unterstützung der Software-Entwicklung hervorbringt, ergänzt durch empirische Studien, die die Güte der Methoden und Werkzeuge evaluieren. Eine spezifische Forschungsmethode für letzteres ist *Action Research*, bei der die ForscherInnen aktiv ihre Methoden in die Praxis bringen [1].

Empirische Forschung im Software Engineering dient also einerseits dazu, Effekte und Phänomene eines Problems zu verstehen, und andererseits dazu, die Güte der Methoden und Werkzeuge für die Problemlösung zu evaluieren.

**Empirische Forschung zur Software-Evolution** sollte deshalb Modelle bereitstellen, um

– die Zukunft (in gewissen Grenzen) vorherzusagen, aufbauend auf Informationen über Vergangenheit, Gegenwart und zukünftige Einflüsse,
– gute von schlechter Evolution zu unterscheiden, d. h. also Vergangenheit und Gegenwart zu bewerten,
– Maßnahmen für gute Evolution zu identifizieren.

Anhand dieser Modelle können neue Ansätze zur Unterstützung der Software-Evolution entwickelt und wissenschaftlich fundiert untersucht werden.

**Herausforderungen.** Diese Ziele bringen typische Herausforderungen mit sich: Da Änderungen untersucht werden sollen, die sehr kontextspezifischen Einflüssen unterliegen und über eine längere Zeit erfolgen, ist es sehr aufwendig, *repräsentative Untersuchungsobjekte* zu finden und daraus entsprechende Informationen abzuleiten. In der Praxis

werden Evolutionsprobleme zumeist erst a posteriori analysiert. Software und ihre Änderungen sind aber in vielen Firmen nicht gut dokumentiert und werden, falls vorhanden, oft nicht offengelegt. Da Softwareentwicklung an vielen Stellen heuristisch geprägt ist, sind weiterhin die menschlichen Einflüsse (durch EntwicklerInnen und NutzerInnen) prägend. Diese können aber nachträglich nur schwer identifiziert werden. Dies bedeutet dass eine systematische wissenschaftliche Analyse mit empirischen Verfahren oft eingeschränkt ist. Mehr Informationen lassen sich über Open-Source-Software gewinnen. Diese wird aber in einem speziellen Kontext entwickelt, der sich von dem Kontext industrieller Softwareentwicklung unterscheiden kann [11]. Dies erschwert es, repräsentative Aussagen über Phänomene zu machen und allgemeingültige Modelle zu erstellen.

Eine weitere Herausforderung ist es, die *Bewertung der Evolution* von der Bewertung der Software zu trennen. Ist Evolution immer dann gut, wenn die resultierende Software gut ist? Die Kriterien für Güte von Software müssen sich den Änderungseinflüssen anpassen. Das würde nahelegen, Software auf die Zukunft hin zu optimieren. Demgegenüber propagieren agile Ansätze Offenheit für Änderungen und häufige Releases (http://www.agilealliance.org/the-alliance/the-agile-manifesto/the-twelve-principles-of-agile-software/). Daher ist die Änderbarkeit in diesen Ansätzen also ein sehr wichtiges Kriterium für die Güte einer Software und des Entwicklungsprozesses.

Ebenso schwierig ist es, die Bewertung der Evolution von der Bewertung der entwickelnden Organisation zu trennen. Ist Evolution immer dann gut, wenn die Organisation effektiv und effizient arbeitet? Das würde nahelegen, die Entwicklungs- und Evolutionsprozesse von Software zu optimieren. Das Gleichgewicht von Artefaktqualität und Prozessqualität wird im Software Engineering schon lange diskutiert. Diese Diskussion fokussiert aber typischerweise auf Entwicklung und nicht auf Evolution. Es ist also nötig, für Evolution eigene Bewertungsmaßstäbe zu entwickeln.

In Bezug auf die evolutionsunterstützenden Maßnahmen ist zu berücksichtigen, dass deren zielgerichteter Einsatz Kosten/Nutzen-Bewertungen erfordert [3]. Diese lassen sich aufgrund der vielen Spezifika einer Software und ihres Kontexts nur schwer erstellen und verallgemeinern. Weiterhin lassen sich unterschiedliche Maßnahmen nicht direkt vergleichen. Im Kontext von Software-Evolution kommt hinzu, dass sich die Kosten/Nutzen-Bewertung angesichts von immer neuen Technologien und Trends kontinuierlich ändert. Es ist also nötig, die Maßnahmen *in der Praxis in unterschiedlichen Projekten über einen längeren Zeitraum* zu evaluieren, bevor eine Aussage über ihre Kosten und Nutzen möglich ist. Ebenso sind Beobachtungen über einen längeren Zeitraum – also oft mehrere Jahrzehnte – nötig, um Evolutionsphänomene zu identifizieren, zu messen und zu bewerten, sowie Vorhersagen zu überprüfen. Der Aufwand dafür ist allerdings sehr hoch und setzt eine langfristige Finanzierung voraus, die in der Wissenschaft selten gegeben ist.

Insgesamt ist festzustellen, dass die für empirische Forschung typischen Themen wie Repräsentativität, Bewertungsmaßstäbe und hohe Kosten in Bezug auf Software-Evolution durch die Notwendigkeit, einen längeren Zeitraum zu untersuchen, verstärkt werden. Dies zeigt sich insbesondere auch in der Rolle des Kontexts: Einerseits ist es sehr wichtig, den Kontext von Software-Evolution zu erforschen, um den Einfluss auf Software-Evolution zu verstehen. Andererseits ist es aber schwierig, diesen Kontext systematisch zu erfassen, insbesondere auch weil er sich in einem längeren Zeitraum häufig ändert.

Im Folgenden berichten wir von der Sicht der ForscherInnen aus dem SPP 1593 auf diese Herausforderungen.

## Bestandsaufnahme im SPP 1593

Im SPP 1593 werden eine ganze Reihe von empirischen Methoden verwendet. Letztendlich sind es unsere Beobachtungen und Erfahrungen mit dem SPP, die die eigentliche Motivation für diesen Artikel lieferten. Um unsere subjektiven Eindrücke methodisch zu unterlegen, haben wir eine „Bestandsaufnahme“ im SPP 1593 durchgeführt. Ziel war es, die empirische Methodik, die im SPP verwendet wird, hinsichtlich ihrer Ausprägungen sowie etwaiger Probleme zu beleuchten. Weitere Teile dieses Artikels stützen sich auf diese Ergebnisse.

Konkret haben wir eine Online-Umfrage durchgeführt und alle MitarbeiterInnen, ProjektleiterInnen und KoordinatorInnen des SPPs dazu

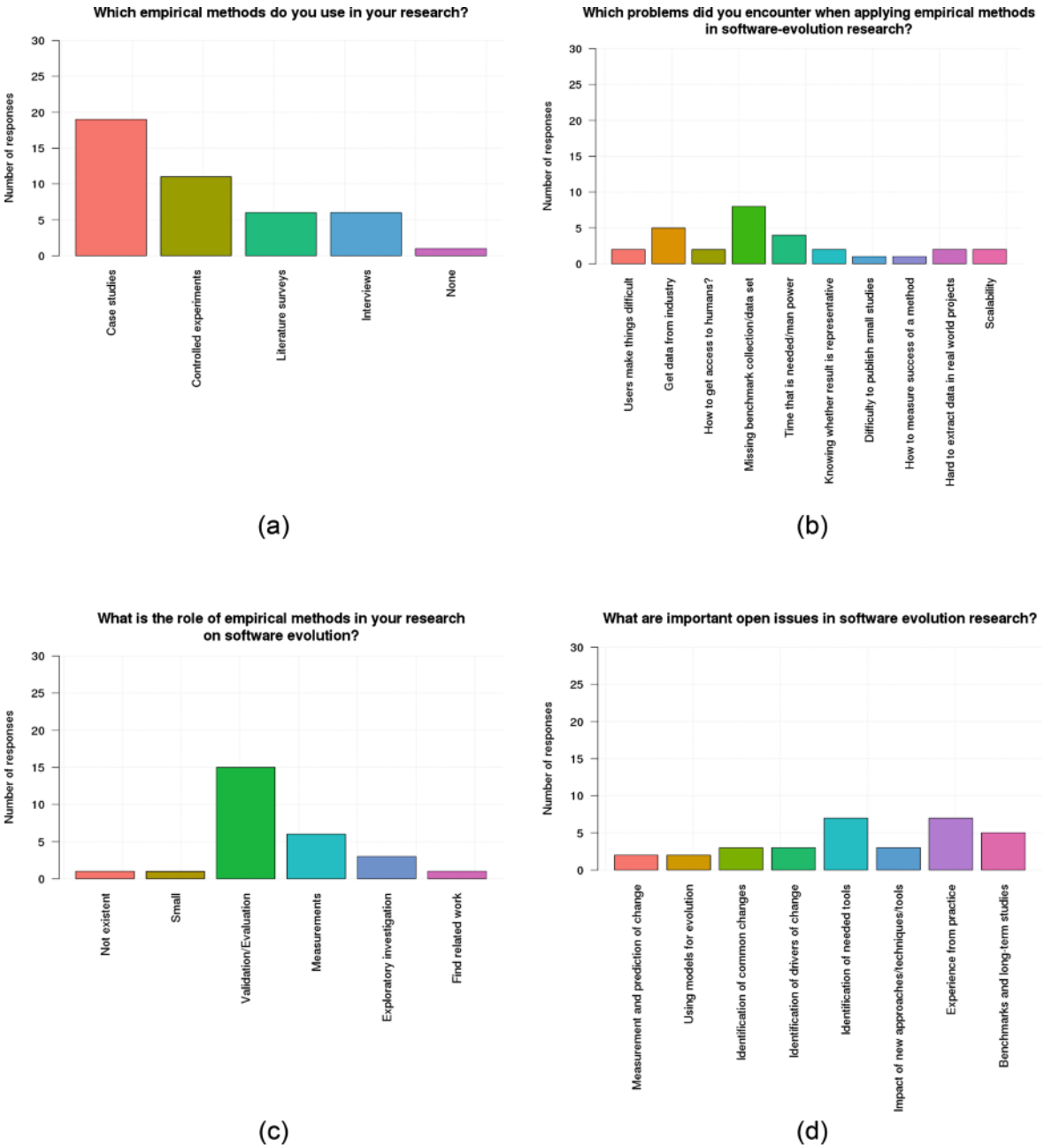


***Abb. 1 Ergebnisse der Umfrage (Auszug)***

eingeladen. Die Einladung wurde per E-Mail verteilt sowie auf einem SPP-Treffen im Oktober 2014 angekündigt. Insgesamt haben sich 20 Personen beteiligt.

Neben einigen Fragen zur Demografie (Erfahrung, Rolle im SPP etc.), enthielt die Umfrage vier Kernfragen (hier verkürzt und auf Deutsch wiedergegeben):

(a) Welche empirischen Methoden werden im Projekt verwendet?
(b) Welche Probleme traten und treten bei der Anwendung dieser Methoden auf?
(c) Welche Rolle spielen diese Methoden im Projekt?
(d) Was sind offene Fragen bzgl. empirischer Methodik in der Software-Evolutionsforschung?

Die Freitext-Antworten der TeilnehmerInnen wurden kategorisiert und sind in Abb. 1 dargestellt.

Obwohl eine Analyse der gesammelten Daten in Tiefe und Breite sicher außerhalb des Rahmens dieses Artikels liegt, möchten wir trotzdem einige interessante Beobachtungen herausgreifen: Wie aus Abb. 1a hervorgeht, spielen kontrollierte Experimente eine wichtige Rolle als Forschungsmethode innerhalb des SPP 1593. Dies ist insofern überraschend und erfreulich, als dass Forschung im Software Engineering traditionell nicht in großem Maße auf strikten empirischen Methoden beruht. Fallstudien waren und sind das Mittel der Wahl. Das ist sicher oft zweckmäßig, kann aber allein nicht Fragen aller Art, insbesondere bzgl. einer Generalisierbarkeit, verlässlich beantworten. Weiterhin ist in Abb. 1b zu beobachten, dass das Fehlen einheitlicher Benchmarks als ein Kernproblem empirischer Forschung im Bereich Software-Evolution im SPP 1593 angesehen wird. Dies lässt auf einen gewissen Reifegrad in diesem Forschungsfeld schließen, da einheitliche und weithin akzeptierte Benchmarks ein wesentliches Mittel zur kritischen Prüfung und Konsolidierung von individuellen Forschungsergebnissen sind (wozu eine ganze Reihe von individuellen Forschungsergebnissen erst einmal vorhanden sein muss). Ein Ergebnis, das weniger überrascht, ist sicher, dass häufig empirische Methoden zur Evaluierung von vorgeschlagenen Ansätzen und Werkzeugen eingesetzt werden, wie aus Abb. 1c ersichtlich ist. Obwohl die Evaluierung ein zentrales Anwendungsgebiet von empirischen Methoden ist, legt dies nahe, dass eventuell empirische Methoden zur Erforschung der Phänomene von Software-Evolution unterrepräsentiert sind oder dass es zu wenig Forschung in dieser Richtung gibt. Schlussendlich ist in Abb. 1d zu beobachten, dass auch Software-Evolutions-Forschung mit klassischen Problemen wie der mangelnden Verfügbarkeit von industriellen Daten zu kämpfen hat. Daneben stehen aber gerade auch langfristige Studien und die Evolutions-Phänomene wie z. B. Fragen nach den Charakteristika und treibenden Kräften von Evolution im Vordergrund, die von der Forschungsgemeinschaft gemeinsam beantwortet werden müssen.

## Forschungsvision

Die Umfrage bestätigt also die am Anfang dieses Beitrags herausgearbeitete Notwendigkeit, die Ziele und die Herausforderungen von empirischer Forschung zu Software-Evolution. Im Folgenden stellen wir unserer Meinung nach wichtige Ausprägungen solcher Forschung vor, die aus detaillierten empirischen Post-mortem/factum-, In-vivo- und In-vitro-Studien Erkenntnisse für verbesserte Methoden und Techniken zur Software-Evolution ableitet. Wir schließen mit einem Plädoyer, diese verstärkt zu fördern.

**Post-mortem/factum-Studien.** Bei den *Post-mortem/factum*-Studien sollten vor allem vorliegende Softwareentwicklungsartefakte aus realen Projekten untersucht werden. Diese Untersuchungen werden heute schon durchgeführt und basieren auf Korrelationsanalysen über die Entwicklung spezifischer Artefakte im Softwareentwicklungsprozess, die Aufschluss über die Evolution geben, wie z. B. die Entwicklung des Softwareprogramms in Relation zu den Fehlermeldungen der NutzerInnen im Issue Tracker. Probleme, die bei den *Post-mortem/factum*-Studien gelöst werden müssen, sind die Verfügbarkeit und Qualität der Daten sowie die Erstellung der Werkzeuge zur automatischen Analyse der Artefakte. Letzteres wird insbesondere im Forschungsumfeld *Mining Software Repositories* vorangetrieben [9]. Zukünftig sollte weiterhin dieser Forschungszweig dazu genutzt werden, zusätzlich zu den gängigen *Korrelationsanalysen* auch präzise *Handlungsempfehlungen* vorzuschlagen. Darüber hinaus eignen sich *Post-mortem/factum*-Studien dazu, grundsätzliche Erkenntnisse über Phänomene und Modelle, also *Gesetzmäßigkeiten und Theorien zu Software-Evolution* zu ermitteln.

Zum Erkenntnisgewinn werden derzeit die Artefakte oft aus frei zugänglichen Software-Repositories entnommen. Diese enthalten zumeist die Änderungen im Quellcode, teilweise auch Fehlerinformationen und mehr. Die Probleme, die sich dabei ergeben und die es in zukünftiger Forschung zu lösen gilt, beruhen auf der diversen Natur der verschiedenen Artefakte. Jede Domäne hat andere Dokumente (z. B. Anforderungsdokumente, Dokument zur Ressourcenplanung und Testmethodik, Zertifizierungsdokumente für sicherheitskritische Anwendungen). Diese müssen bei Projekten in realistischer Größe für reproduzierbare, empirische Analysen miteinander

verknüpft werden, und es muss ein vereinheitlichtes Verständnis für die Modellbildung generiert werden.

**In-vitro-Studien.** Aufbauend auf den Erkenntnissen und Theorien der *Post-mortem/factum*-Studien, die grundsätzliche Korrelationen und Zusammenhänge offenbaren, sollte in den *In-vitro*-Studien ein *Ursache–Wirkungszusammenhang* ermittelt werden. Dies entspricht einem experimentellen Forschungsansatz [13]. *In-vitro*-Studien sollten, wie der Name schon offenbart, im Labor oder in einer abgeschlossenen Umgebung ablaufen. Dabei sollte eine Experimentierumgebung erstellt werden, die gezielt einen der Faktoren (z. B. das Vorhandensein von Verfolgbarkeits-Beziehungen zwischen den Artefakten) isoliert und die Auswirkung auf einen abhängigen Faktor (z. B. Wartungsaufwand bei der Evolution des Softwaresystems) untersucht. Üblicherweise werden solche Studien mit eher kleineren Softwaresystemen und unter Nutzung von Studierenden als ProbandInnen durchgeführt. Dies führt zu einer Generalisierungsschwäche (geringe externe Validität), bringt jedoch durch die Ermittlung von Wirkungszusammenhängen (hohe interne Validität) einen weiteren Erkenntnisgewinn gegenüber den *Post-mortem/factum*-Studien.

**In-vivo-Studien.** Zur detaillierten Ermittlung der *grundsätzlichen Gesetzmäßigkeiten der Software-Evolution* bedarf es abschließender *In-vivo*-Studien. Hier wird die Software-Evolution nicht pur auf Basis der Entwicklung von Artefakten (*post mortem/factum)* oder im Labor (*in vitro*) untersucht, sondern in realen Softwareentwicklungsprojekten. Hierbei bieten sich vor allem Längsschnittstudien an, welche die Evolution von Softwaresystemen auch unter Berücksichtigung von realen Softwareentwicklungsprozessen und der SoftwareentwicklerInnen selbst untersuchen. Da besonders langlebige Softwaresysteme eine Herausforderung darstellen, sollten diese Längsschnittstudien über mehrere Jahre (>10 Jahre) erfolgen und den gesamten Lebenszyklus betrachten. Dabei lässt sich die Forschungsmethode der Längsschnittstudien gezielt mit dem bereits erwähnten *Action-Research*-Ansatz unterfüttern, bei dem gewonnene Erkenntnisse der Software-Evolutionsforschung in den Entwicklungsprozess injiziert werden. Bei den *Post-mortem/factum*- und *In-vitro*-Studien gibt es bereits eine Vielzahl von existierenden Arbeiten [8, 9]. Im Gegensatz dazu werden *In-vivo*-Längsschnittstudien nur wenig durchgeführt und kaum dokumentiert. Dies ist nach Ansicht der AutorInnen ein großes Manko der Software-Evolutionsforschung, da dadurch viele Erkenntnisse nicht abschließend bestätigt sind. Auf der anderen Seite würde eine verstärkte Durchführung von *In-vivo*-Längsschnittstudien auch neue Forschungsfragen aufwerfen, die gezielt aus der Praxis in die wissenschaftliche Arbeit kommen.

## Schlusswort

Zusammenfassend plädieren wir

- für einen Ausbau der Forschung zu Software-Evolution,
- unterfüttert durch empirische Forschung zu Phänomenen und Wirksamkeit von Methoden und Werkzeugen.
- Dabei sollten insbesondere langfristige *In-vivo*-Studien durchgeführt werden.

In anderen Disziplinen wie der Medizin, sind langfristige Studien fest etabliert. Sie sollten trotz des hohen Aufwands auch für das Software Engineering anerkannt und gefördert werden. Nur so kann gewährleistet werden, dass erfolgversprechende Methoden und Modelle für die Software-Evolution, wie sie aktuell im SPP1953 entwickelt werden, umfassend evaluiert werden und dann eine nachhaltige Umsetzung in die Praxis finden.

## Danksagung

Wir danken der DFG und den Mitgliedern des SPP 1593 für ihre Unterstützung.

## Literatur

1. Avison D, Lau F, Myers MD, Nielsen PA (1999) Action research. Commun ACM 42(1):94–97
2. Basili VR, Caldiera G, Rombach DH (1944) The Experience Factory. In: Marciniak JJ (ed) Encyclopedia of Software Engineering, vol 1. John Wiley & Sons, Inc., pp 469–476
3. Biffl S, Aurum A, Boehm BW, Erdogmus H, Grünbacher P (2006) Value-based software engineering. Springer, New York
4. Endres A, Rombach DH (2003) A Handbook of Software and Systems Engineering. Empirical Observations, Laws and Theories. Pearson Education Publishing
5. Engels G, Goedicke M, Goltz U, Rausch A, Reussner R (2009) Design for Future* Legacy-Probleme von morgen vermeidbar? Informatik Spektrum 32(5):393–397
6. Godfrey MW, German DM (2008) The Past, Present, and Future of Software Evolution. In: Frontiers of Software Maintenance. IEEE Computer Society, pp 129–138

7. Hevner AR, Ram S, March ST, Park J (2004) Design Science in Information Systems Research. MIS Q 28(1):75–105
8. Kemerer CF, Slaughter S (1999) An Empirical Approach to Studying Software Evolution. IEEE T Softw Eng 25(4):493–509
9. Kagdi HH, Collard ML, Maletic JI (2007) A Survey and Taxonomy of Approaches for Mining Software Repositories in the Context of Software Evolution. J Softw Maintenance 19(2):77–131
10. Mens T, Gueheneuc Y-G, Fernández-Ramil J, D'Hondt M (2010) Guest Editors' Introduction: Software Evolution. IEEE Softw 27(4):22–25
11. Paulson JW, Succi G, Eberlein A (2004) An Empirical Study of Open-Source and Closed-Source Software Products. IEEE T Softw Eng 30(4):246–256
12. Runeson P, Höst M, Rainer A, Regnell B (2012) Case Study Research in Software Engineering. Guidelines and Examples. Wiley, Hoboken, NJ
13. Tichy WF (1998) Should Computer Scientists Experiment More? Computer (Long Beach, CA) 31:32–40
14. Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslén A (2012) Experimentation in Software Engineering. Springer, Berlin New York