INSIGHT Computer Science

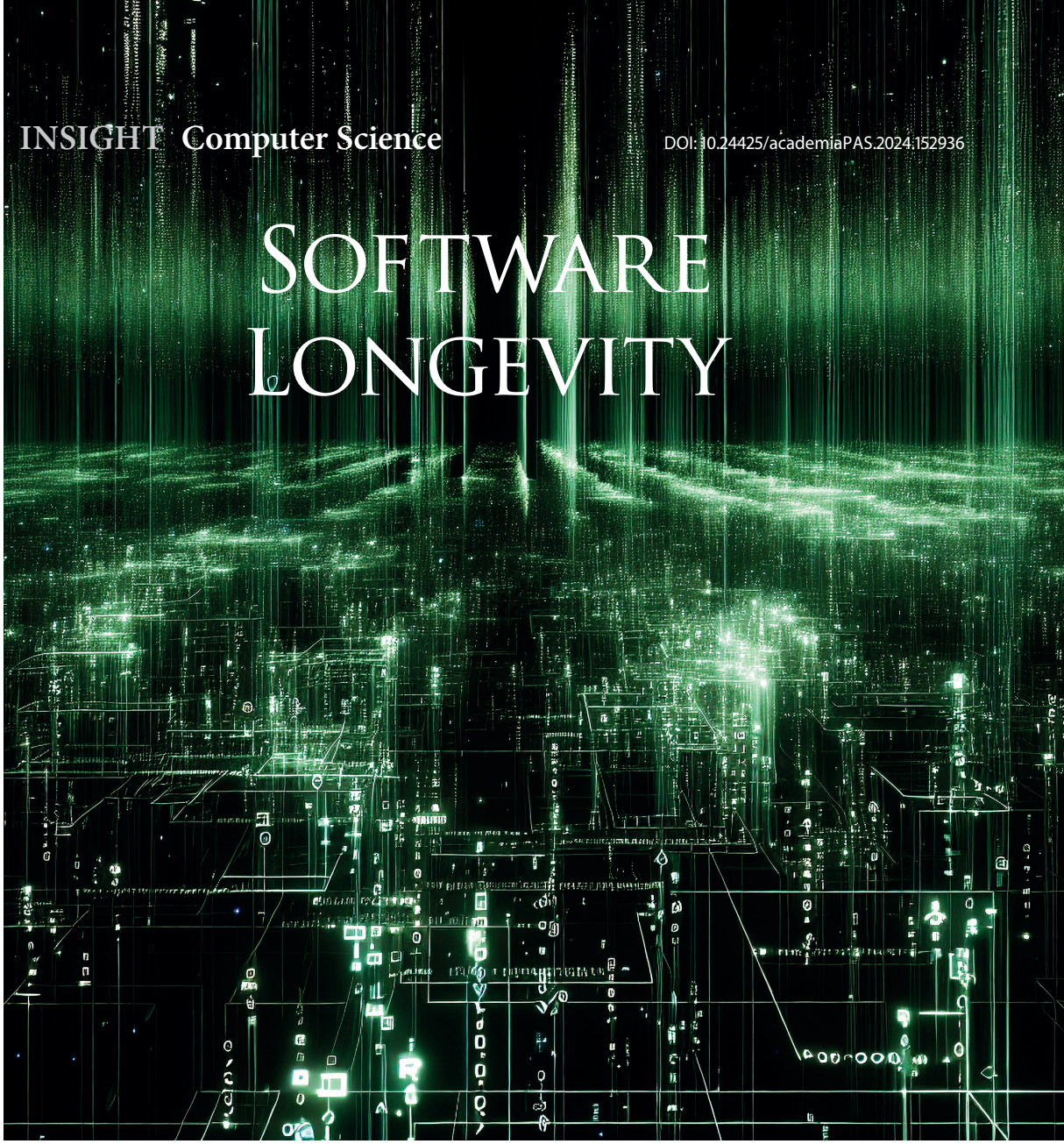
DOI: 10.24425/academiaPAS.2024.152936

Software Longevity

Bartosz Walter, PhD, DSc

works at the Institute of Computing Science at the Poznań University of Technology and the Poznań Supercomputing and Networking Center. His research focuses on software evolution and code maintenance methods. He is passionate about sharing science with the public.

bartosz.walter@cs.put.poznan.pl



SHUTTERSTOCK GENERATOR AI

Just like living organisms, software undergoes "aging" and requires regular maintenance to stay functional and adaptable.

Bartosz Walter

Institute of Computing Science,
Poznań University of Technology
Poznań Supercomputing and Networking Center

As the well-known saying goes, health is something we don't appreciate until we lose it. Old age, reflecting the cumulative effects of past events, neglect, and illness, often brings this wisdom into sharp focus – albeit usually a bit too late. Can we take this advice to heart earlier, when it can still make a difference? We all certainly try in one way or another. In the pursuit of lasting health and youth, we aim to delay aging through exercise, healthy eating, and maintaining a so-called healthy lifestyle, or by undergoing various rehabilitative therapies. While these measures cannot eliminate aging, they can help us stay active for many years. In this sense, our health depends on our ability to combat aging. As long as we have the strength and resources to continue, we remain healthy and capable.

Aging and the natural decline of function do not apply to just human health; they are universal to all living organisms. But interestingly, this approach to the concept of "health" can also be applied beyond humans and other organisms, to include objects and the products of human activity. For instance, buildings are designed and constructed for specific

purposes (residential, functional, decorative), but throughout their lifetimes, they are often adapted, renovated, or reconstructed. A building remains functional as long as it meets current needs or can be modified to do so. When its adaptation becomes impossible, the building "dies," in a sense, much like living organisms do.

While the process of aging varies widely, one constant remains: health depends on the ability to adapt to change. In biology, this means resilience against disease, the capacity to regenerate worn-out organs, and proactive measures against aging. In fields such as construction engineering or mechanics, this means preventing wear and tear while maintaining the ability to adopt new functions or retain existing capabilities.

Aging Programs

This concept of aging can also usefully be applied in what might be an unexpected domain: computer software development. At first glance, it might seem absurd: sure, a computer can age, a printer can break down, or a monitor can wear out, but how can a program – essentially a sequence of instructions executed by a machine – grow "old"? Software can grow outdated or ill-adapted to contemporary requirements, but there's more to it than that. Software itself is also subject to degradation over time, in a process known as *software aging*.

Every program inevitably contains bugs that need fixing, and users frequently request enhancements or new features. Additionally, software often glitches after the operating system updates, requiring adjustments to restore functionality. Every modification to a program, no matter for what reason, introduces a potential for damage – whether through accumulating fixes, deviations from the original design, or the risk of new bugs emerging while existing ones are being addressed. This means that every program change comes with a certain "bill to pay," including both direct costs (coding, testing, and deployment) and indirect costs (every fix or new feature makes future modifications more challenging or even impossible). To counteract this, developers must smooth out changes and eliminate "scars," which requires additional effort. This is similar to rehabilitation after invasive surgury: while it may be optional in some cases, skipping it may often compromise one's quality of life or even longevity.

Just like one has to take care of one's own wellbeing, the health of buildings and software also require a certain upkeep. Neglecting this maintenance results in something called *technical debt* – a term programmers use for unresolved issues that accumulate over time. Like financial debt, technical debt doesn't disappear on its own; it demands repayment, often in the form of additional maintenance hours. While manageable in moderation, excessive technical debt can lead to a "debt spiral," where most resources get spent on maintaining old rather than developing new software. At some point, the program becomes unable to adapt to new changes, and ultimately "dies."

Preventative Screening

To stay confident about your personal health, you could choose to undergo regular advanced tests to monitor for any and all signs that your health balance is shifting. However, this approach is expensive and not especially practical for most people. Alternatively, one might resign to the idea that aging is inevitable and simply accept it. A more pragmatic approach lies in finding a middle ground – one that balances the benefits of proactive monitoring with the realities of cost and convenience. In medicine, this is where screening tests come into play. These tests focus on identifying affordable and relatively simple-to-detect indicators, even if they aren't always perfectly specific and can sometimes produce misleading results.

> Much like human health conditions, not every issue in software programs demands an immediate response.

The same applies to software: instead of subjecting it to constant analysis and thorough review, we can focus on certain specific symptoms known to be likely to lead to trouble and a rapid increase in technical debt. Currently, we are aware of several dozen such symptoms. They concern, for example, module complexity, opaque structures, highly repetitive code, or a strong dependency on other components that could "infect" neighboring modules with errors. But how do we actually know that such features are risky? We use the same methods as modern evidence-based medicine: collecting data on the effectiveness of specific "treatments" and analyzing it statistically. As a result, we can predict future challenges with reasonable accuracy – though always with statistical caveats.

Much like human health conditions, not every issue in software programs demands an immediate response. In most cases, simply monitoring their status is sufficient to prevent unpleasant surprises. This approach enables effective management of a program's health at a relatively low cost, ensuring it continues to serve and satisfy its users for years to come. ■

Further reading:

Król K., *Code Decay, Software Erosion, and Software Entropy*. 2019. homeproject.pl

Parnas DL., Software Aging. *Proceedings of the 16th International Conference on Software Engineering (ICSE '94)*, 1994.