When legacy code turns into senescent code: Assessing software aging and its implications

Philip König (SBA Research, PKoenig@sba-research.org), Kevin Mallinger (SBA Research, KMallinger@sba-research.org), Alexander Schatten (SBA Research, ASchatten@sba-research.org)

Research Paper, Cluster [3]: Digitalization and methods for technology assessment: new approaches

In the last decades it became clear that, more often than not, software engineers were focused on delivering new features and in the process generated highly complex interacting layers and modules of software over long periods of times, where very old legacy code interacts in complex ways with new code. Various studies imply that software aging is a real phenomenon whereupon continuous execution of programs leads to a gradual build-up of errors and overall degradation of performance. Strange and unclear behaviour emerges from interactions of new and old modules, which in the worst case manifest itself in crashes, errors and other unwanted responses. This has especially drastic consequences for critical infrastructure networks like power grids or medical software, where conventional practices for error detection like controlled shutdowns and reboots are seldomly an option. Invaluable for early detection of such issues are non-invasive methods which would serve as detectors to assess when, for example, a software module begins to show first symptoms of developing aforementioned and similar problems. While first technology assessment methodologies concerning software aging have been developed none drew inspiration from the natural sciences, where, especially in the last few decades, biogerontology – the science of processes of aging and its consequences – has begun to pick up serious steam. By abstracting similar processes in biology and computer science the fundamental problems and their solutions can be analyzed and then transferred from one to another. Recent discoveries showed that chronological age is not always an accurate variable to define a cell's or an organism's biological age, as different organisms and even different cell types within those organisms age at different rates. Cancer cells even evolved mechanisms to periodically rejuvenate themselves and thus became quasi-immortal. Similarly, it is important to not use chronological age as the only parameter to determine if and when code ages, as very old code that continuously gets maintained and cautiously ported to new platforms might exhibit less signs of software aging than chronologically younger modules which were dragged along by techniques like wrapping. Therefore different biological strategies and parameters for true age assessment will be analyzed, such as the Hayflick limit, which describes how dividing cells count how often they already multiplied by periodically shortening dedicated parts of their chromosomes. After a certain amount of divisions a critical length is reached and a suicide program is initiated to reduce the risk of becoming a cancer cell. There has also been recent work on cell rejuvenation by autophagy, a process where under certain conditions cells start a maintenance program which detects and removes unnecessary or dysfunctional components. Such modes of operation will be compared to those of proposed software rejuvenation techniques and potentially supplement and augment those methodologies. Thus ultimately this paper aims to achieve a transfer of knowledge from biology to computer science to capture systemic processes of software aging that foster the improvement of the long-term functionality of cyber-physical systems and their resilient design.