The Danger of Architectural Technical Debt: Contagious Debt and Vicious Circles is an article written inside of the Software Architecture (WICSA), 2015 12th Working IEEE/IFIP Conference on 16 July 2015 by Antonio Martini and Jan Bosch, professors at Chalmers University of Technology.

This article reviews the experiment conducted by the authors in their study of ADT. ADT or Architecture Technical Debt (ADT) is errors in code that creates architecture intended to support the business goals of the organization. An example, given in the article, of an ADT is presence of structural violations. More generally, Technical Debt, is the process of applying the easier solution or "responsiveness" in the short term goal instead of applying the better overall solution or long term "responsiveness". Technical Debt is analogous to debt itself in that taking loans to pay for short term debts can lead to more debt in the future paid which has to be paid with interest. The focal point of this article is to examine different teams and answer three questions: which ADT items caused the most damage, what are the consequences of these items and what leads to ADT.

The paper utilized numerous case studies to understand the effects of ADT. They broke down the collection of data into three phases: in the first phase they interviewed a variety of different roles which culminated into a one-day workshop involving 40 people from all the different case studies. The author did phase 1 to better understand what was architecture practices were currently be utilized in the industry. The second phase again utilized interviews of different roles. Each interview was followed by a process which would identify architectural inconsistencies with high impact damage (effort). The third phase included more interviews where the authors identified to the team the architectural weakness and asked the teams to provide improvements. Finally, as an additional step the authors organized two plenary workshop with teams outside of the experiment conducted to strengthen results.

The authors broke down the architectural problems into five different categories: Dependency violations and unawareness, Non-uniformity of patterns and policies, Code duplication (non-reuse), Temporal properties of inter-dependent resources, and Unidentified non-functional requirement. What makes these categories so important is that all of the architectural problems can be broken down to into one of these categories and as a result many of the architectural problems encountered are likely to have similar solutions if they are part of the same Taxonomy.

Beyond the categories, the authors attention turned towards a pattern of events which create loops and can lead to acclamation of effort interest to no end. This relationship is identified as vicious circles and what makes them so important is because ADT items by themselves have a fixed interest cost, but vicious circles keeps adding ADT over time; leading to an increase in the ADT effects.

Vicious Circles can be broken down in to three categories: Lack of uniformity and familiarity, Hidden ATD, not Completed Refactoring and Time Pressure, Contagious ATD. Why this is important is because vicious circles can create large scale problems for a team and being able to identify can lead to faster solutions and have production time for the team. These three categories for vicious cycles can also help teams understand the cycles and prevent them from occurring in their code in the first place.

The paper makes an important point in their discussion section towards the end. In it they state "we see the need, in future research, to identify architectural solutions that would avoid the phenomenon, but also stop the spreading once the contagious ATD item would be identified." What makes this so important is that more and more companies are becoming places where different parties collaborate and complete. As a result, identifying an ADT item in even a small part of the code can lead to dangerous consequences for the product and company as a whole. Being finding solutions that not only solve the ADT item but correct it everywhere in the system would prove to be vital in the coming age.

Lastly it is important to note that the paper concedes that is has limitations. As the paper state, the measurements taken during the experiment are not exact and are rather to illuminate practices that threaten software development. It important for the reader to understand that while the paper does a good job in identifying some of the different ADT items and vicious cycles present in the code, there is a chance that certain one are still not identified. As a result, a programmer should be use the identifications of ADT and vicious cycle categories along with other coding practices to ensure a quality product.