InsighTD Survey
Investigating causes and implications of Technical Debt

Types of technical debt and their indicators

List of TD types (Alves *et al.*, 2018):

Type	Definition	Situations where debt items can be found
Design Debt	Refers to debt that can be discovered by analyzing the source code and identifying violations of the principles of good object-oriented design.	- Violations of the principles of good object-oriented design; - Some types of code smells; - Complex classes or methods; - Excessive design complexity.
Code Debt	Refers to the problems found in the source code (poorly written code that violates best coding practices or coding rules) that can negatively affect the legibility of the code making it more difficult to maintain.	- Unnecessary code duplication and complexity; - Bad style that reduces the readability of code; - Over-complex code.
Architecture Debt	Refers to the problems encountered in product architecture, which can affect architectural requirements. Usually, architectural debt could be the result of sub-optimal upfront solutions, or solutions that become sub-optimal as technologies and patterns become superseded, compromising some internal quality aspects, such as maintainability.	- Violation of modularity; - Complex architectural behavioral dependencies; - Architectural compliance issues; - System-level structure quality issues; - Non-uniform usage of architectural policies and patterns; - Lack of handling interdependent resources; - Lack of addressing non-functional requirements; - Implementation of immature architecture techniques.
Test Debt	Refers to issues found in testing activities that can affect the quality of those activities.	- Insufficient test coverage; - Lack of tests (e.g., unit tests, integration tests, and acceptance tests); - Deferred testing; - Lack test case planning.
Documentation Debt	Refers to the problems found in software project documentation.	- Missing documentation; - Inadequate documentation; - Outdated documentation; - Incomplete documentation.
Defect Debt	Refers to known defects, usually identified by testing activities or by the user and reported on bug tracking systems, that the development team agrees should be fixed but, due to competing priorities and limited resources, have to be deferred to a later time.	- Postergated decisions on fix defects, bugs, or failures found in software systems.

Infrastructure Debt	Refers to infrastructure issues that, if present in the software organization, can delay or hinder some development activities. Such issues negatively affect the team´s ability to produce a quality product.	- Delaying an upgrade or infrastructure fix; - Outdated components of an application's development environment; - Sub-optimal configuration of development-related supporting tools.
Requirements Debt	Refers to tradeoffs made with respect to what requirements the development team needs to implement or how to implement them. In other words, it refers to the distance between the optimal requirements specification and the actual system implementation.	- Requirements that are only partially implemented; - Requirements that are implemented but not for all cases; - Requirements that are implemented but in a way that doesn't fully satisfy all the non-functional requirements.
People Debt	Refers to people issues that, if present in the software organization, can delay or hinder some development activities.	- Late hire.
Build Debt	Refers to issues that make the build task harder, and unnecessarily time consuming.	- The build process can involve code that does not contribute to value to the customer; - If the build process needs to run ill-defined dependencies, the process becomes unnecessarily slow; - Manual build process.
Process Debt	Refers to inefficient processes, e.g. what the process was designed to handle may be no longer appropriate.	- Manual processes with the potential to be automated accrue interest in terms of manual labour costs.
Automation Test Debt	Refers to the work involved in automating tests of previously developed functionality to support continuous integration and faster development cycles.	- Lack of automated testing.
Usability Debt	Refers to inappropriate usability decisions that will need to be adjusted later.	-
Service Debt	Refers to the inappropriate selection and substitution of web services that lead to mismatch of the service features and applications' requirements. This kind of debt is relevant for systems with service-oriented architectures.	- Selection or replacement of web service.
Versioning Debt	Refers to problems in source code versioning, such as unnecessary code forks.	- Unnecessary code forks.

InsighTD Survey
Investigating causes and implications of Technical Debt

List of TD indicators (Alves *et al.*, 2016):

Indicators	TD Type
Violation of Modularity	*Architecture Debt*
Software Architecture Issues	
Betweenness Centrality	
Augmented Constraint Network (CAN)	
Pairwise-Dependency Relation (PWDR)	
Index of Package Changing Impact (IPCI)	
Index of Package Goal Focus (IPGF)	
Structural Dependencies	*Architecture Debt / Build Debt*
Structural Analysis	*Architecture Debt / Design Debt*
Build Issues	*Build Debt*
Code without Standards	*Code Debt*
Slow Algorithm	
Multithread Correctness	
Code Metrics (not specified)	*Design Debt / Code Debt*
Automatic Static Analysis (ASA) Issues	
Code Smells	
Grime	*Design Debt*
Software Design Issues	
Low External / Internal Quality	
Afferent / Efferent Couplings (AC / EC)	
Depth of Inheritance Tree (DIT)	
Referential Integrity Constraints (RICs)	
Uncorrected Known Defects	*Defect Debt / Test Debt*
Insufficient Comments in Code	*Documentation Debt*
Lack of Documentation	
Comments (hack, fixme, is problematic, ...)	
Documentation Issues	
-	*Infrastructure Debt*
-	*People Debt*
-	*Process Debt*
Requirement Backlog List	*Requirement Debt*
Selection/Replacement of Web Service	*Service Debt*
Lack of Automated Testing	*Test Automation Debt*
Incomplete Tests	*Test Debt*
Defects Deferred	
Insufficient Code Coverage	
Lack of Test Case Documentation	
Lack Test Case planning	
-	*Usability Debt*
Unnecessary code forks	*Versioning Debt*

References

Alves, N. S., Mendes, T. S., de Mendonça, M. G., Spínola, R. O., Shull, F., & Seaman, C. (2016). Identification and management of technical debt: A systematic mapping study. *Information and Software Technology*, *70*, 100-121.

Rios, N., Mendonça, M.G., and Spínola, R.O. 2018, A tertiary study on technical debt: Types, management strategies, research trends, and base information for practitioners, Information and Software Technology, Volume 102, Pages 117-145, ISSN 0950-5849, https://doi.org/10.1016/j.infsof.2018.05.010