# Tech Debt

Technical Debt is a programming concept that is also known as code debt, it applies to any type of code that is not written to standards. The debt is looked at as the cost to bring the code up to standards. As part of my recent Scrum Master certification course (yes I am a certified scrum master now) the concept of technical debt came up. Technical debt is something that I have been involved with arguing with between developers and DBAs on many occasions, but until the Scrum Master certification course, the concept of technical debt never really stuck for me. In the past when technical debt was argued, it was often times used to write off those areas of code that developers and DBAs just don't enjoy working on, but there is more to it than that. Analyzing technical debt is like getting a report card on the work you do, or on the systems you have to maintain.

The concept of Technical Debt applies closely with Agile software development, but also applies to database development. There are many tools available to report on technical debt for C#, PHP or other programming languages, but there isn't much available to manage the T-SQL technical debt or general database design technical debt on SQL Server.

Some of the traditional technical debt causes are:

- Poor design.
- Lack of coding standards.
- Different conflicting coding standards over time.
- A rush to get things done rather than focusing on stability and sustainability.
- Non refactoring code as it grows.
- Evolving code or environments to just make things work, rather than to make things work well.
- many more…

Reducing technical debt will lead to a more sustainable environment.

## Example of Technical Debt in SQL Server

> Consider a project that has been using SQL Server over the last 10 years which has evolved from SQL Server 2000, and is currently running on SQL Server 2008 with a plan to move to SQL Server 2012. There were practices with SQL Server 2000 that were considered best practices, or the right way of doing things that aren't the case any more. For instance in SQL Server 200o using TEXT columns was the norm for anything that was larger than 8000 bytes that wouldn't fit into a VARCHAR column. In SQL Server 2005 Microsoft introduced the concept of VARCHAR(MAX) to go beyond the 8000 byte limit (along with NVARCHAR(MAX) and VARBINARY(MAX) ). It has been rumored that TEXT columns would be going away since SQL Server 2008, but they are still supported in SQL Server 2008 R2. SQL 2012 has the TEXT column on the official deprecated features list with the statement of
> "Use varchar(max), nvarchar(max), and varbinary(max) data types."

In this example where Microsoft has announced that the TEXT column type will be going away at some point in the future, the use of the TEXT column type is considered technical debt. Just using that column will cause (based on the deprecated feature list from Microsoft) your code to break at some point in the future. TEXT columns still work in SQL Server 2012 even though it is deprecated, and in the CTP1 early release of SQL Server 2014 they still work, but since Microsoft has announced that they are deprecated who knows for the future.

Technical debt is one of those things that is up for much debates based on your specific coding standards. Anything that analyzes technical debt needs to allow specific overrides bases on your local coding standards. For instance some people argue that using spaces instead of tabs for indentation in your T-SQL code is the right way, and others argue that tabs are the only way to do proper indentation. For this specific case there are many pros or cons to either side of the argument, the key is coding standards. If your coding standard is one way or the other,

you are in better shape than if it is not defined.

## Making Changes

Lets say you have a stored procedure in your SQL Server database that needs to be changed because business needs have changed, it could be for any reason. If that stored procedure is low on technical debt, it is likely that you will be able to adjust or modify it to do the things you need it to do, and you may be able to do that with relatively few errors. If that stored procedure was instead high in technical debt, standards weren't followed, just plain bad code, it would not be so easy to modify, and it would probably introduce more problems to try and change it. Technical debt is handy to use to assess the amount of work involved in making changes to your database. As a DBA or developer, we often times blame the people who worked on the system before us, technical debt analysis of your SQL Server allows you to find that specific areas that were done poorly by your predecessors (or you) and to focus on how to correct them.

Here is a quote from twitter that really sums up the concept of fragile code associated with technical debt:


## Technical Debt Components

- Object Selector
- Settings Dialog
- Technical Debt Object Viewer
- Technical Debt Overview

## Monetary Value of Technical Debt

The monetary value of technical debt in SQL Server is based on a formula using the complexity or the cost of specific debt items, and dividing that by the amount of technical cost that can on average be fixed in a day, then multiplying that by the cost of a DBA / Database Developer for one day. These numbers may vary depending on the skill set of your team, and the cost of a DBA or Database Developer for a day.

Database Health Reports includes a technical debt analysis tool. Give it a try for FREE!.

## Related Links

- Technical Debt Overview
- Database Health Download Page