CS5351 Software Engineering
2024/24 Semester A
Technical Debt

Dr W.K. Chan

Department of Computer Science

Email: **wkchan@cityu.edu.hk**

Website: **http://www.cs.cityu.edu.hk/~wkchan**

Outline

- What is Technical Debt
- Identify Technical Debt
- Manage Technical Debt
- Card Game for Managing Technical Debts
- Managing Technical Debts in Practice

Technical Debt in Media

Forbes

5,132 views | May 30, 2016, 09:33pm

Technical Debt: The Silent Company Killer

Falon Fatemi

Much like a poor diet, technical debt starts with good intentions but ends in frustrating system failures. But unlike unhealthy meals, technical debt is nearly unavoidable: Today's agile developers focus on delivering software quickly to solve a market need. Developers who over-architect or over-code software only delay launch and waste money.

So while shortcuts and assumptions help developers iterate quickly, they also cause software to accrue technical debt. Left unaddressed, this debt bogs down systems, crashes software, and causes multi-month delays in product releases.

CISION

Innovation Paralysis: How Legacy Systems and Technical Debt Are Choking Enterprise Growth

PR Newswire
September 10, 2024 • 4 min read

Hidden Cost of Technical Debt and Legacy Systems

According to Unqork's survey of 500 business and technology leaders across financial services, insurance, healthcare, and the public sector, 85% reported that time spent maintaining legacy systems hampers their ability to launch new solutions. This ongoing battle with technical debt and legacy systems exhausts vital resources, strangling most organizations' capacity to innovate.

Technical Debt in Media

IT Brief AUSTRALIA

How tech debt from legacy solutions will impact business' bottom line

By Contributor, 16 Sep 2021

Article by Pure Storage VP & field CTO Mark Jobbins

Public and private-sector organisations are making massive investments in digital transformation to adapt to the new economy. However, many of these investments rely on building new applications on legacy IT architecture. This technology approach is leading to rising levels of tech debt, impacting organisational performance and increasing cyber-risk.

Organisations that take a half-hearted approach to digital transformation may find themselves with rising levels of tech debt and systems that don't adapt to business changes. A recent McKinsey survey found that tech debt currently accounts for 40% of IT balance sheets. Sixty per cent of chief information officers (CIOs) reported increasing spending on tech debt.

Technical debt occurs because of interdependencies between new systems and legacy IT architecture that often require hard coding. This creates an expensive IT infrastructure for the organisation, which is often challenging to maintain. Clear warning signs of tech debt include hard coding and financial arrangements that masquerade as cloud solutions.

Some tech debt is unavoidable because it is a cost of doing business as part of the transition from old to new technology. However, too much tech debt has severe financial, operational, and security implications for businesses.

Tech debt increases the total cost of ownership of IT for organisations and directly impacts competitiveness because it is prone to security vulnerabilities. It is also unlikely to scale and adapt at the same speed as new technology solutions due to a heavy reliance on manual processes.

Cyber-criminals may find it much easier to target organisations that have higher levels of tech debt due to the likelihood of greater security gaps in IT architecture from disparate systems and a lack of IT integration.

Just like financial debt, organisations cannot afford to sustain tech debt in the long term. It is a critical business issue rather than purely a technology problem. As such, tech debt must be addressed in terms of its broader organisational impacts over the longer term.

Technical Debt in Media

Inside Tech's $2 Trillion Technical Debt

By Shane Tews

AEIdeas

June 03, 2024

Technical debt, the accumulation of shortcuts and compromises in software systems, is an issue across industries with consequences ranging from system failures and security breaches to hindered innovation. The Wall Street Journal reports it was the cause of 13,000 canceled Southwest Airlines flights during the 2022 holiday season and numerous high-profile cyberattacks on Google, Apple, and Microsoft. The cost of addressing this debt stands at $1.52 trillion, while cybersecurity incidents, operational failures, and maintenance of outdated systems amounts to $2.41 trillion annually in the US.

Technical Debt in Media

InfoWorld

Home > Software Development

How to minimize new technical debt

With best practices and a commitment to not let technical debt grow, developers can make a solid business case, especially when staffing and money are tight.

By Isaac Sacolick

Mon, Sep 26, 2022 Newsweek

U.S. World Tech & Science Culture Autos Rankings Health Life Opinion Experts Educatior

EXPERTS

Privacy Debt: The Hidden Scalability Killer

Privacy debt most often takes the form of missing policies, insufficient procedures, a lack of awareness in the organization and, ultimately, a lack of visibility into the personal information processed by a company.

ROSS SAUNDERS, PRIVSEC DIRECTOR, BAMBOO DATA CONSULTING
ON 8/29/22 AT 12:01 PM EDT

Technical debt prevention practi development

John Kodumal, CTO and cofounder of LaunchDarkly is inevitable in software development, but you can c proactive: establishing policy, convention, and proc cost of reducing debt over time. This is much health other work and trying to dig out from a mountain of

Kodumal recommends several practices, such as "es policy and cadence for third-party dependencies, using workflows and automation to manage the life cycle of feature flags, and establishing service-level objectives."

To help reduce the likelihood of introducing new technical debt, consider the following best practices:

- Standardize naming conventions, documentation requirements, and reference diagrams.
- Instrument CI/CD and test automation to increase release frequency and improve quality.
- Develop code refactoring as skills and patterns practiced by more developers.
- Define the architecture so developers know where to code data, logic, and experiences.
- Conduct regular code reviews to help developers improve coding practices.

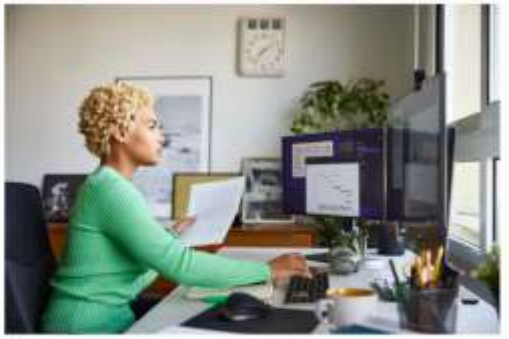
Forbes

INNOVATION

Measuring And Managing Technical Debt

Ken Knapton Forbes Councils Member
Forbes Technology Council COUNCIL POST | Membership (Fee-Based)

Dr. Knapton is currently the CIO at Progrexion. His expertise is in big data, agile processes and enterprise security.



Engineers spend 33% of their time dealing with technical debt. Multiple studies estimate the average organization wastes 23%-42% of their development time on technical debt. In those same studies, CIOs

Technical Debt

- Technical debt (TD)
  - is a metaphor originally proposed in 1992 [1]. The metaphor is that doing things in a "quick and dirty" way creates a technical debt (TD). Like a financial debt, TD incurs interest payments, which come in the form of the extra effort that developers must dedicate in future development because of this quick and dirty design choice.
  - is a design or construction approach that is expedient in the short term but that creates a technical context in which **the same work will cost more to do later than it would cost to do now** (including increased cost over time) [2].
- Examples
  - "Guys, we don't have time to dot every *I* and cross every *T* on this release. Just get the code done. It doesn't have to be perfect. We'll fix it after we release."
  - "We don't have time to reconcile these two databases before our deadline, so we'll write some glue code that keeps them synchronized for now and reconcile them after we ship."

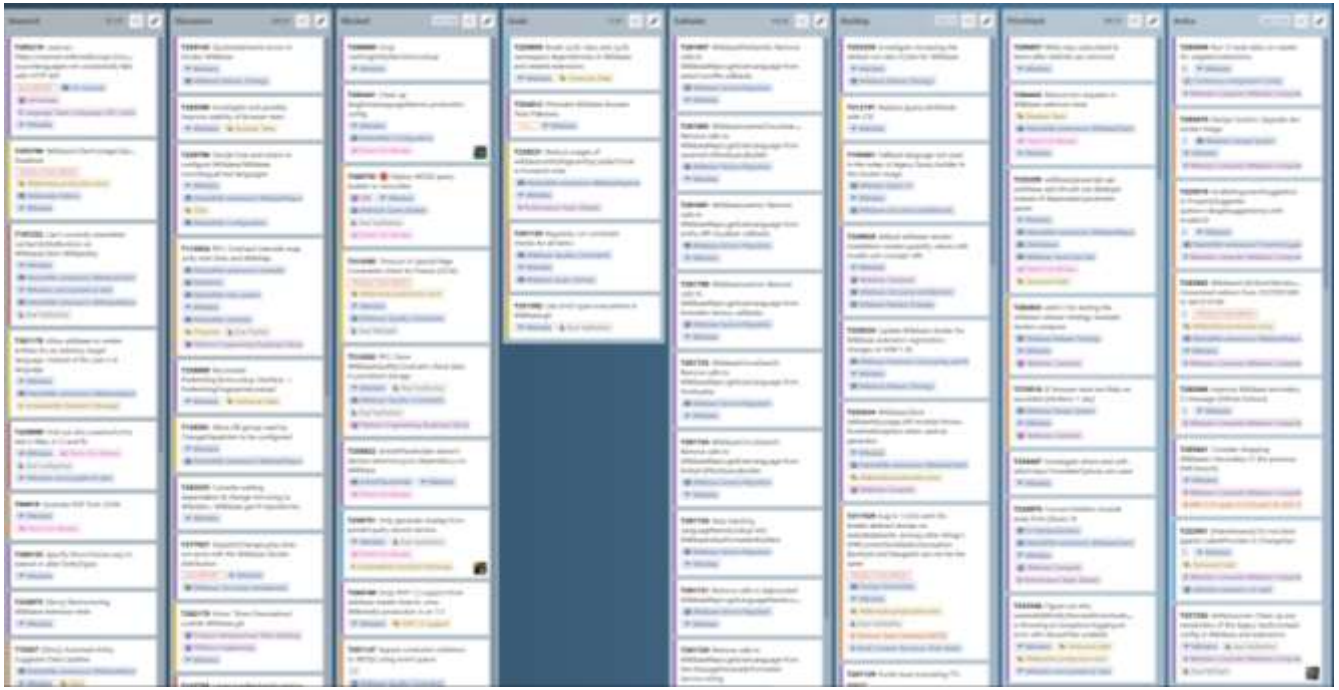
Examples

Technical debt in task board

Debt	Business Justification	Sprint Incurred	Estimate	Work item	Commitment	Note
Variable names preceded with "email" even though emain is only part of their use	Confusion factor is not currently there but will be in the future	sprint 2	1 week	Change variable manes to more meaningful names	sprint 3	DONE
Duplicate codes with displays in Remedy Controller and Favorites view	Meet early May release schedule, willing to accept cost of double maitenance for now	sprints 3-4	3 weeks	New file to contain common methods for remedy display, change existiong code to conform	Release 1.1 - expected out in Italy 2010	Servicing this debt may push the SEARCH function out to a later release
The same method handles both designating and undesignating a favorite	Too risky to do right before the customer milestone demo	sprint 6	2 weeks	Bust out undesignate into separate method, add logic to parts of code to know which method to invoke	Sprint 7, possibly sprint 8	

https://qarea.com/blog/how-to-reduce-tech-debt-a-practical-experience-guide

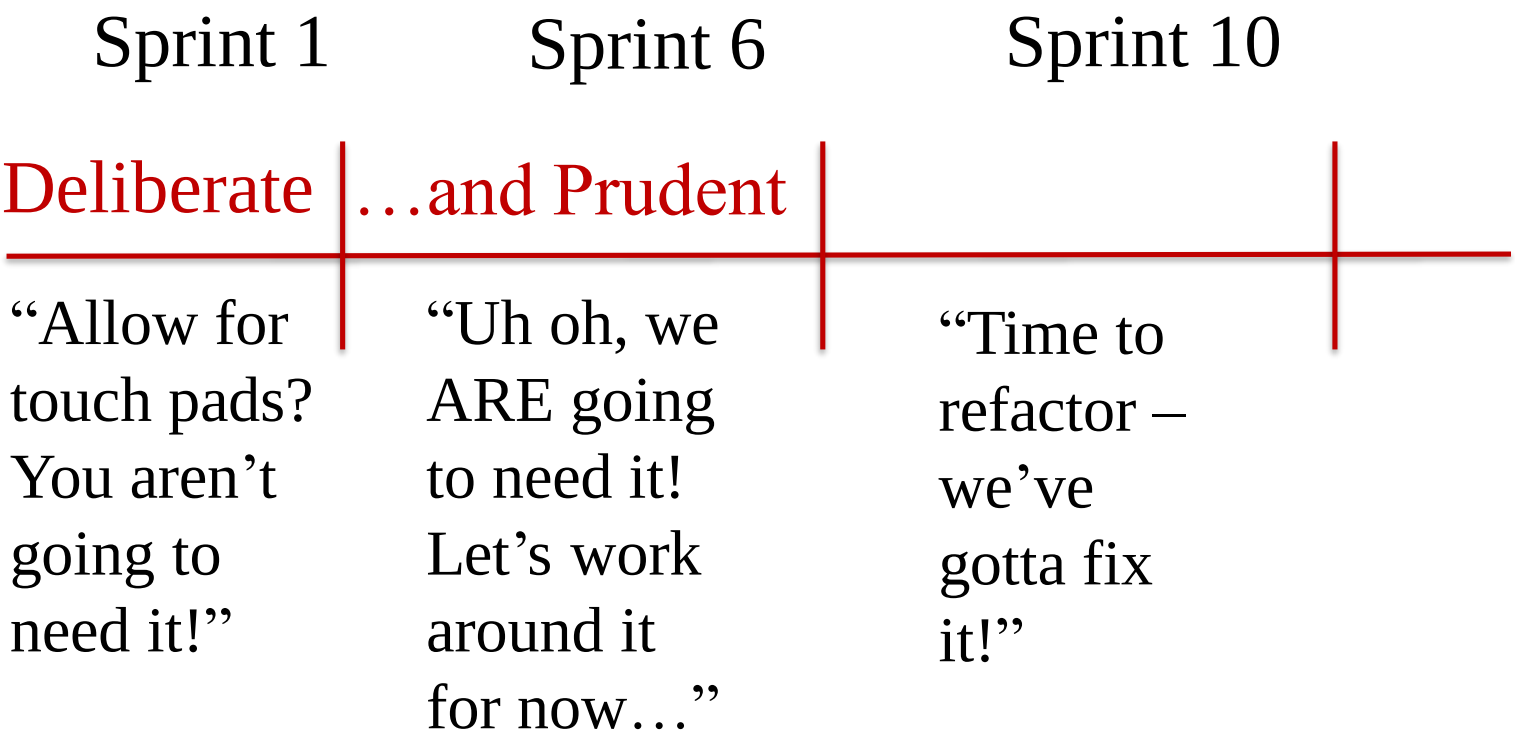
https://theagilepmo.blog/2016/12/30/technical-debt-visualised-using-cost-of-delay/



https://addshore.com/2021/06/tackling-technical-debt-big-and-small-in-wikidata-and-wikibase/

Technical Debt
incur, then pay interest, finally pay back

Typical scenario:



Examples of "Interest Payments" on Technical Debt

- *e.g.*, Lack of unit tests on legacy code causes new development, testing, and debugging to take longer
- *e.g.*, Overly simple design does not readily support changes in the environment, and seemingly simple environment changes require massive code rework
- *e.g.*, High product support cost for a buggy system (non-software interest payment)
- *e.g.*, Brittle system means each bug-fix introduces unintended side effects (e.g., new bugs), so each bug-fix becomes a multibug-fix project
- *e.g.*, Bug reports are so frequent that time spent fixing bugs in the production system prevents any work on new functionality
- *e.g.*, Overly lengthy edit-compile-debug times due to poor development environment (non-code cause of technical debt)

Counterexamples of Technical Debts

- Many kinds of delayed or incomplete work are not technical debt:
  - feature backlog, deferred features, cut features

- In general, if “the same (development) work ***will not*** cost more to do later than it would cost to do now”, then it is not a technical debt.

Technical Debts in Practice [8]

- Ernst et al. surveyed 1831 software engineers and architects found:
  - 31% stated TD as an implicit part of their backlog
  - 25% stated TD as an explicit part of their backlog
- TD items commonly are not present in the official sprint backlog; they are often documented in quite ad-hoc managed *shadow backlogs* (i.e., “unofficial”, e.g., private notes of the team or a member, comments in source code commits).
- A sprint backlog includes a mix of different items (e.g., new features, bugs/defects, and improvements). TD is often grouped as Improvements.
- When TD is present in the software, **the only significantly effective way of reducing TD is to refactor the software**.

Impacts of Technical Debts

- In some large organizations, **development time** dedicated to managing Technical Debt is **substantial**
  - an average of 25% of the overall development
  - Some said 40%-50%

- Let’s look at how companies view technical debts (next slide)

Company View

- Many companies tend to *trade*
  - their longer-term ability to produce new releases frequently, with high quality, quick feedback and small effort

in exchange for

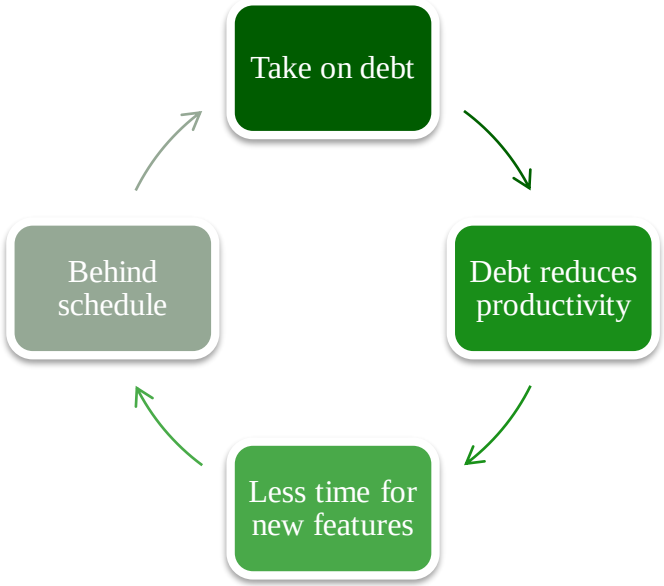
  - short term features that might give them a quick fix but slow them down in the long run.

- This strategy makes these companies vulnerable to their competitors

Two Sides of the Same Story with a Company [2]

- Business View
  - Shorten time to market
  - Preserve startup investment
  - Delay development expense
- Business staff tend to be ***optimistic*** about the benefit and the costs. These attitudes are often ***unconscious***.

- Technical View



- Technical staff tends to be ***pessimistic*** about the benefit and the costs. These attitudes are often ***unconscious***.

Technical Debt and Its Basic Management

- Technical debts have been classified by dimension:
  1. **Type** (e.g., design, testing, or documentation debt),
  2. **Intentionality** (intentionally or unintentionally),
  3. **Time horizon** (short or long term), and
  4. **Degree of focus**.

Use them to judge a technical debt

- (Popular) Technical Debts management idea
  - **Identify** a technical debt and **track** it through product backlog of the software project. **Discuss** them.
  - E.g., put the following into the backlog: “We don't have time to reconcile these two databases before our deadline, so we'll write some glue code that keeps them synchronized for now and reconcile them after we ship.”

Dimension 1:
Types of Technical Debt

- Technical Debt is not just a coding problem.
- Refactoring out the code smells **cannot** solve them all.

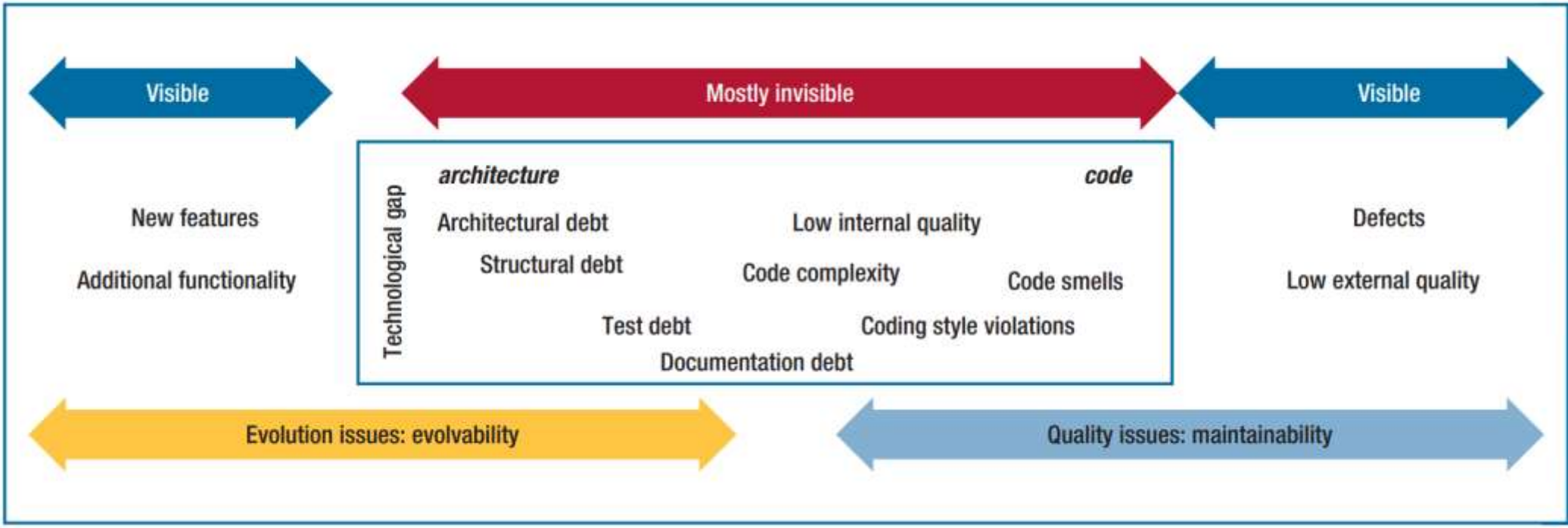


FIGURE 1. The technical debt landscape. On the left, evolution or its challenges; on the right, quality issues, both internal and external. [3]

Dimension 1:
Types of Technical Debt

Table 1
Definition of technical Debt (Li et al., 2015).

TD type	Definition
Requirements TD	"Refers to the distance between the optimal requirements specification and the actual system implementation, under domain assumptions and constraints"
Architectural TD	"Is caused by architecture decisions that make compromises in some internal quality aspects, such as maintainability"
Design TD	"Refers to technical shortcuts that are taken in detailed design"
Code TD	"Is the poorly written code that violates best coding practices or coding rules. Examples include code duplication and over- complex code"
Test TD	"Refers to shortcuts taken in testing. An example is lack of tests (e.g., unit tests, integration tests, and acceptance tests)"
Build TD	"Refers to flaws in a software system, in its build system, or in its build process that make the build overly complex and difficult"
Documentation TD	"Refers to insufficient, incomplete, or outdated documentation in any aspect of software development. Examples include out-of-date architecture documentation and lack of code comments"
Infrastructure TD	"Refers to a sub-optimal configuration of development-related processes, technologies, supporting tools, etc. Such a sub-optimal configuration negatively affects the team's ability to produce a quality product"
Versioning TD	"Refers to the problems in source code versioning, such as unnecessary code forks"
Defect TD	"Refers to defects, bugs, or failures found in software systems"

https://doi.org/10.1016/j.jss.2020.110827

Dimension 1:
Types of Technical Debt

TABLE I
TD ITEMS (NUMBER AND EFFORT DISTRIBUTION) PER TYPE BY PROJECT.

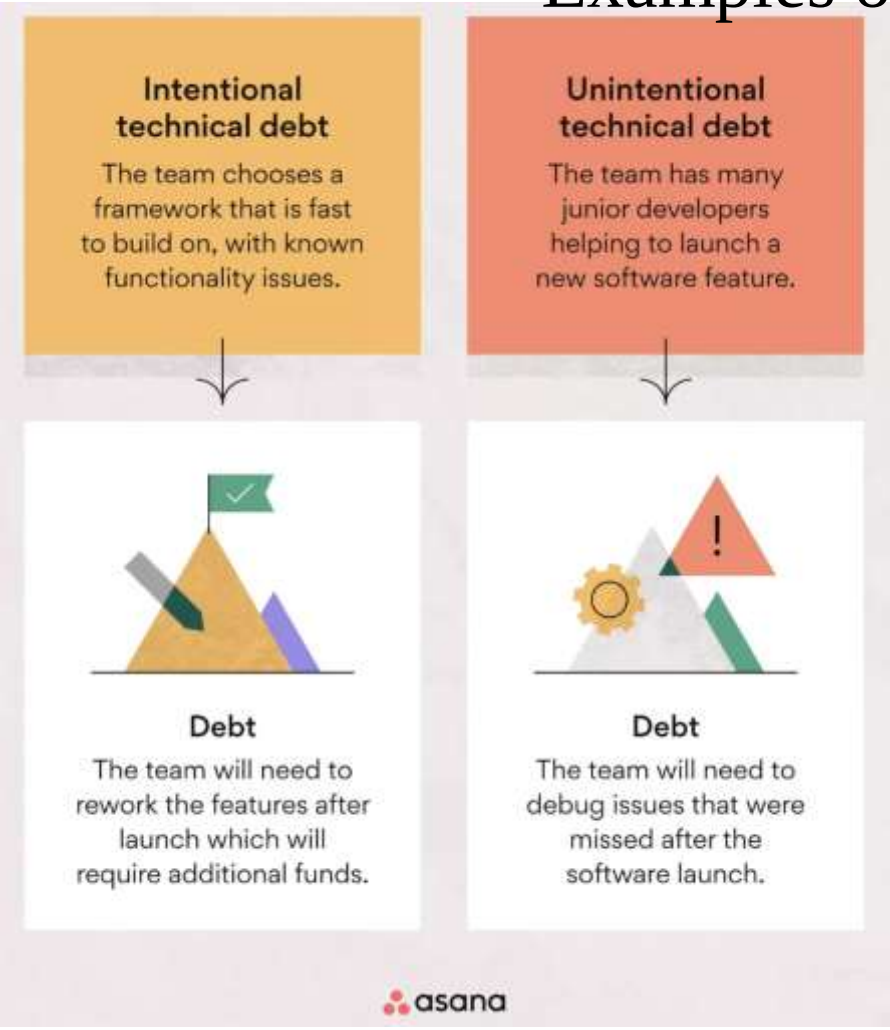
TD Types	Man-hour effort estimate														
	Company-1						Company-2			Company-3			Company-4		
	Project-A			Project-B			Project-C			Project-D			Project-E		
	Item	M/H	%	Item	M/H	%	Item	M/H	%	Item	M/H	%	Item	M/H	%
Architecture	2	1,680	58.8%	1	-	-				4	108	17.7%	1	-	-
Code	11	198	6.9%	5	-	-	4	382	13.7%	6	74	12.1%	2	-	-
Defect	2	32	1.1%	2	-	-	5	373	13.4%	11	175	28.7%	3	-	-
Design					-	-				6	27	4.4%	2	-	-
Doc.	3	292	10.2%		-	-				4	20	3.3%	1	-	-
Infrastructure					-	-	11	885	31.8%					-	-
Process					-	-	2	884	31.7%				1	-	-
Requirement	4	464	16.2%	10	-	-	2	90	3.2%	11	170	27.9%	7	-	-
Service				1	-	-								-	-
Test					-	-				1	5	0.8%		-	-
Usability	4	191	6.7%	5	-	-	3	173	6.2%	5	31	5.1%	3	-	-
Total	26	2,857	100.0%	24	-	-	27	2,787	100.0%	48	610	100.0%	20	-	-

https://doi.org/10.1109/ESEM.2019.8870180

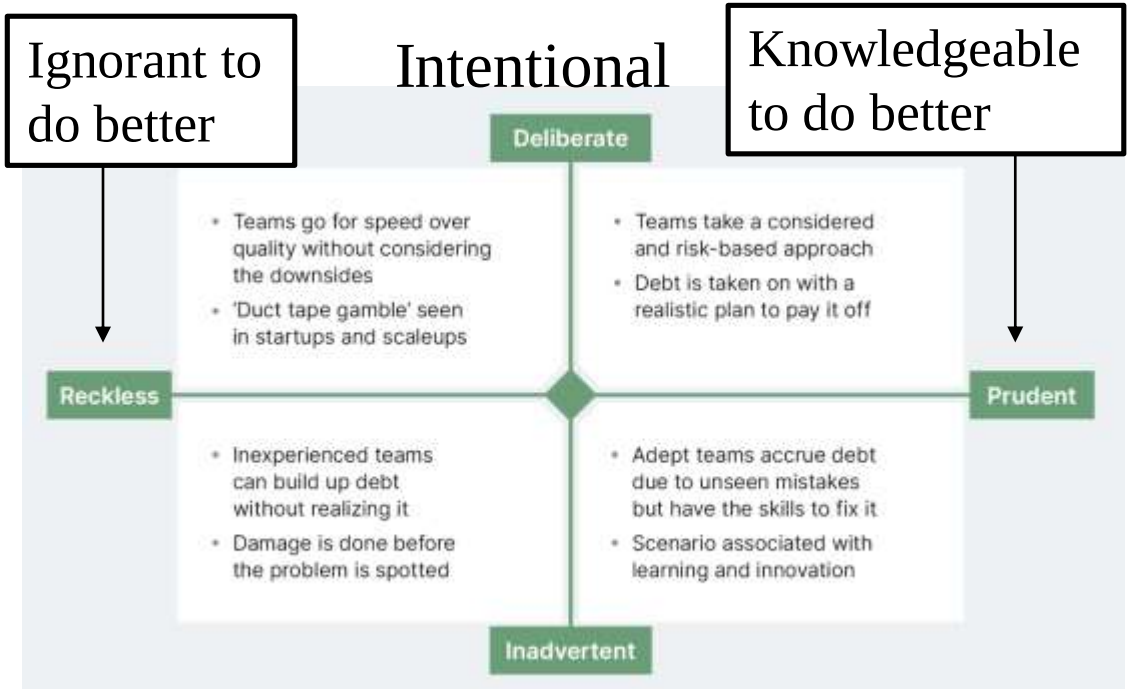
Spread over all development lifecycle phases

Dimension 2:
Intentionality of Technical Debt

Examples on the Web



https://asana.com/zh-tw/resources/technical-debt



See the examples in the next slide

Dimension 2:
Intentionality of Technical Debt

Examples:

- **Prudent and deliberate**: Ship the product quickly and address any issues later. This debt is created when the benefits of a quick delivery outweigh the risks (the stakes in the software project are relatively low).
- **Reckless and deliberate**: Know how to produce the best code but prioritize quick product delivery over producing the best code. It will result in legacy code that is harder to maintain.
- **Prudent and inadvertent**: It occurs when the team aims to write the best code but later discovers a better solution after implementation.
- **Reckless and inadvertent**: It occurs when a team attempts to write the best code without having the necessary knowledge to do so. They may not realize the mistakes they're committing. This can introduce vulnerabilities and maintainability issues.

Dimension 2:
Intentionality of Technical Debt

- Intentional Technical Debt
  - Debt incurred by intention
  - e.g., “If we don’t get this release done, there won’t be a second release.”
  - e.g., “We don’t have time to build in general support for multiple platforms. We’ll support iOS now and build in support for Android, etc., later.”
  - e.g., “We didn’t have time to write unit tests for all the code we wrote the last 2 months of the project. We’ll write those after we release.”

Dimension 2:
Intentionality of Technical Debt

- Unintentional Technical Debt
  - Debt incurred unintentionally due to low quality work.
  - e.g., A junior programmer writes bad code
  - e.g., A newcomer/contractor writes code that doesn't follow the coding standard
  - e.g., A major design strategy (e.g., decide to use the model-view-controller architecture to connect all major modules in the software) turns out poorly
  - e.g., Your company acquires a company that has a lot of technical debt
  - e.g., A comprehensive refactoring task goes sideways
  - e.g., Honest Mistakes. "I wish we'd known Framework 2.1 would be so much better than Framework 2.0."
  - e.g., Careless Mistakes. "Design? What design? Hardware design?

Dimension 3:
Time Horizon

- Short-Term Debt
  - Usually incurred **reactively**, for tactical reasons
  - e.g., Skipping some integration tests to get a release out the door
  - e.g., "We don't have time to implement this the right way; just hack it in, and we'll fix it after we ship."
  - e.g., Violating coding standards to upload a hotfix
- Long-Term Debt
  - Usually incurred **proactively**, for strategic reasons
  - e.g., “We don’t think we’re going to need to support a second platform for at least 3 years, so our design supports only one platform.”

Dimension 4:
Degree of Focus

- Focused Debt
  - Debts that are intentionally incurred and managed.
- Unfocused debt
  - Debts accumulated when the team has no clear strategy or priority for managing and addressing technical debts

	Unfocused Debt	Focused debt
Prioritization	Not systematic	Systematically identified and managed
Planning	Reactive. No long-term planning strategy	Proactive. Long-term planning exists
Documentation	Poor	Good
Accumulation	Growing and unmanageable	Managed and proactively reduced
Communication	Insufficient within the team	Ongoing within the team

Dimension 4:
Degree of Focus

Examples of Unfocused Debts

- A legacy application with mixed functions and classes, making it hard to understand and change.
- New features are added without unit tests, leading to a fragile codebase that breaks easily.
- Features are merged quickly without proper code review, resulting in low-quality code that needs extensive rework.
- The application uses an old framework with known security issues and no plans for updating, increasing risk.
- A software product has accumulated many features over time, making its codebase bloated and hard to manage.
- The application lags during peak usage, with performance optimizations ignored in favor of adding new features.
- An outdated CRM or ERP system is used that doesn't integrate well with modern applications, leading to duplicated efforts and data issues.

Dimension 4:
Degree of Focus

Examples of Focused Debts

- *Deliberate* shortcuts for time-to-market
- *Temporary* code workarounds
- Code simplicity over complexity
- Deferring non-critical refactoring
- Selectively use legacy solutions
- Inadequate testing for quicker delivery
- Experimental feature in production
- Skipping documentation
- Limited error handling
- Implementing high-priority features while knowing the underlying infrastructure inadequate
- Prototype code in production
- Deferring external library's dependency update
- Use naïve/simplified algorithms or data structure
- Feature cut for meeting deadlines

Identify Technical Debts

Patterns to Increase Technical Debts

- There are some well-known patterns in software development to *increase* technical debts

1. Schedule Pressure
2. Duplication of Code
3. Get it “right” the first time

Patterns to Increase Technical Debts:

1. Schedule Pressure

- When a team is held to a commitment that is unreasonable, they are bound to cut corners to meet management expectations.
  - E.g., creeping scope of new features, change in team composition, late integration

- Solution
  - Use a more flexible planning approach

Patterns to Increase Technical Debts:
2. Duplication of Code

- Many reasons for code duplications
  - Lack of experience on part of the team members
  - Copy-and-paste programming practice
  - Conform to the poor design of existing software
  - Pressure to deliver; guess a schedule
- Solution
  - Use static analysis tool for assistance (see Code for Quality slides)
  - Pair programming
    - To spread the knowledge and improve competence of team members
  - *Either* (a) Repay debts now *or* (b) Track it if replay later,
    - (1) fix it now, (2) add runtime exception to location of technical debts and fix it a few minutes later, (3) add the debt (location, description, potential cost of not fixing it) to the project backlog

Patterns to Increase Technical Debts:

3. Get it "right" the first time

- Opposite of duplication – create a lot of powerful design up-front.
  - We have over-engineered and over-generalized our intended product.
- Costs associated with fixing *keep growing*

Manage Technical Debts

Mange Technical Debts [5]

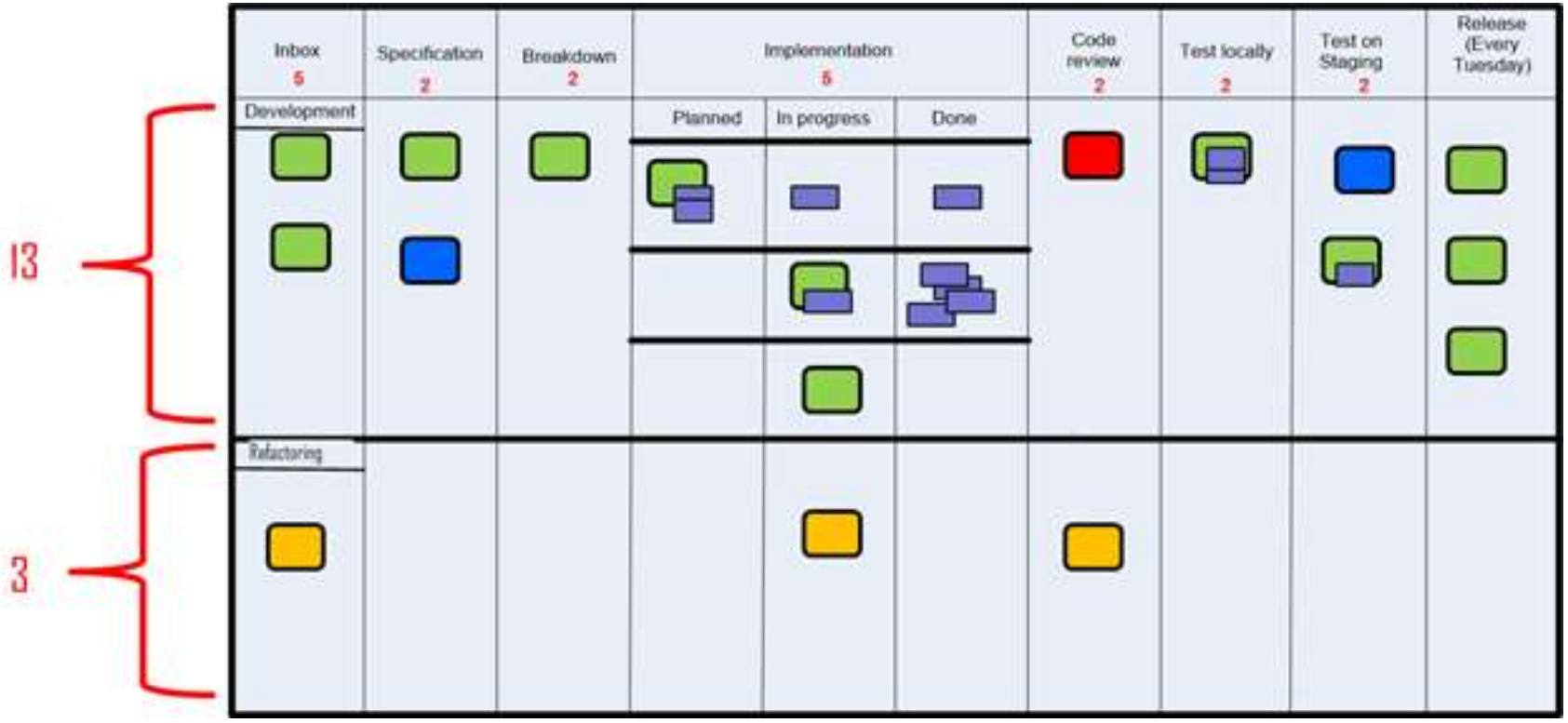
- If we can’t avoid technical debt, we must manage it.
- This means recognizing it, tracking it, making reasoned decisions about it, and preventing its worst consequences.

Methods to Manage Technical Debts:
Tracking Technical Debts [2]

- Tracking technical debts (TD) is necessary to manage them.
  - All “good debt” can be tracked (at least by definition)
  - Log as defects
  - Include in product backlogs
  - Monitor project velocity
  - Monitor amount of rework
- Ways to Measure Debt
  - Total of debt in product backlog
  - Maintenance budget (or fraction of maintenance budget)
  - Measure debt in money, not features, e.g., “50% of R&D budget is nonproductive maintenance work”

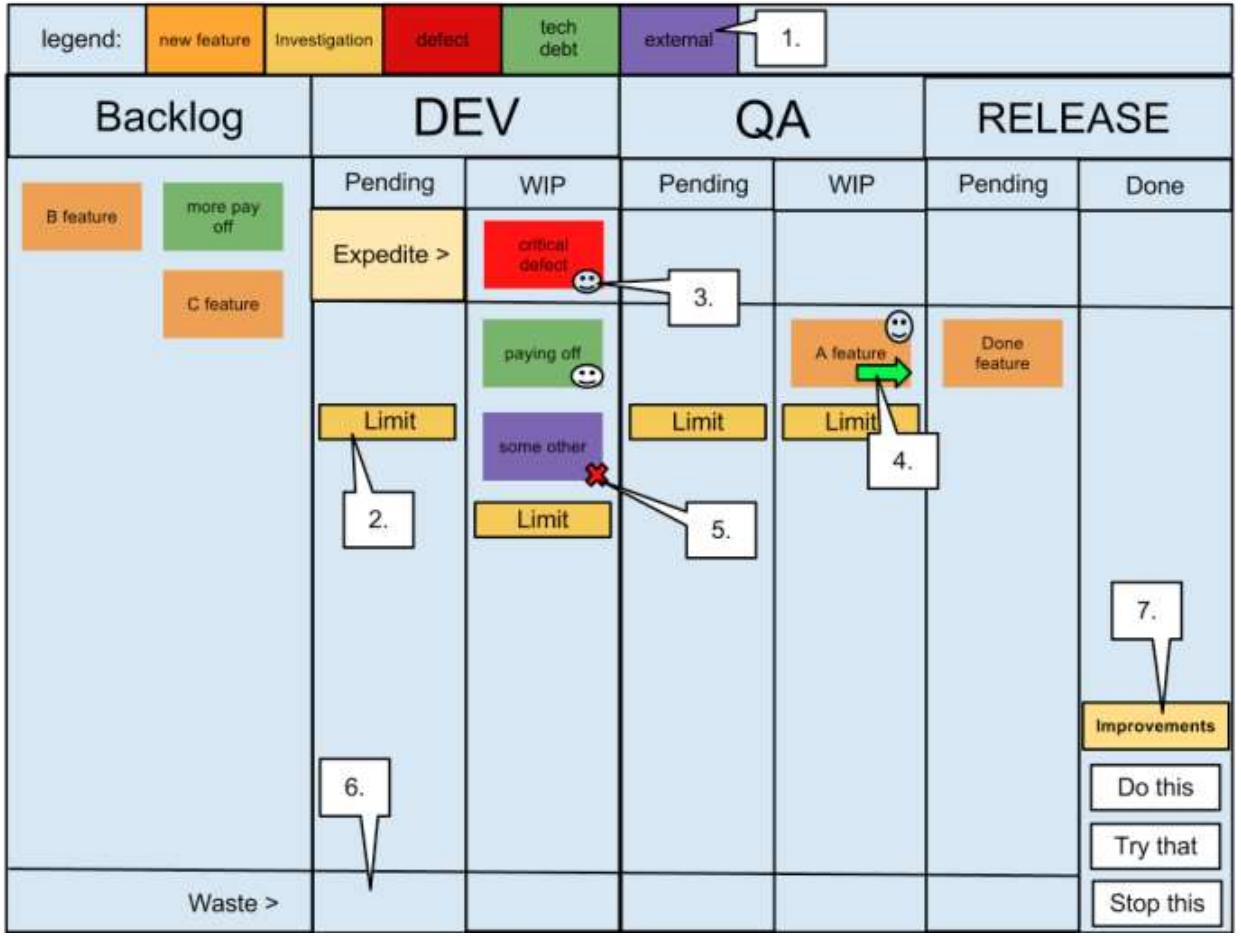
Methods to Manage Technical Debts:
Tracking Technical Debts Visually [6]

- Visualize the technical debts on task boards. The lower section includes three refactoring tasks (i.e., instances of technical debts) to be addressed in a project: one is under code review, another one is “in progress”, and the last one has not been handled at all.



Methods to Manage Technical Debts:
Tracking Technical Debts Visually [6]

- Visualize the technical debts on Kanban board. (for the 1-7, see the next slide)



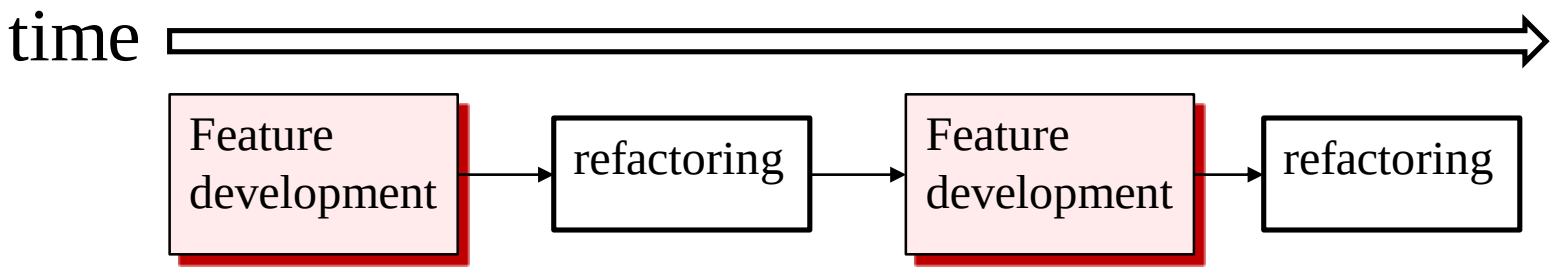
The seven legends on the last slide

- **Legend** – This shows the types of cards (by color) that can be on the board, these include New features, Tech debt, Defects, Investigations (support) and External Issues (e.g. new servers). The purpose of this is to clearly **visualize** the type of work that is being processed
- **Limits** – Each pending and work in progress column has a limit to control the amount of work that can be taken on. These limits are set based on resource levels and other **bottlenecks**, and are intended to encourage the PULLING of work when capacity exists rather than continuing to force more work onto a bottleneck.
- **Avatar** – Each team member has ONE avatar to place on the item they are on, this tells other team members that this card is being worked upon and should be progressing. As we pair on most development, there will often be more than one avatar on each Dev task.
- **Arrows** – Indicate movement of tasks since the last standup, again this is part of the **visualization** of the process. These cards are often ignored at the standup, as we want to focus on the items that are not progressing, such as…
- **Blockers** – We want to minimize blockers as they cause more bottlenecks in the system, so we clearly indicate these items with blocked avatar. An additional requirement to blocking a card is to add the details of WHO and WHY the item is unable to proceed.
- **Waste** – Tasks can be abandoned at any stage in the process, we collect these and review at the weekly **retrospective**.
- **Continuous Improvements** – Also, at the weekly **retrospective** we decide upon three improvement points that we want to tackle for the following week. These are summarized on the board and recalled at the beginning of the morning standup, if we fail to make any progress on an improvement then it may be rolled over to the next week.

Methods to Manage Technical Debts:

Repay Technical Debts

- An effective tactic is to reserve a timeslot on a periodic basis to deal with technical debt.



- Longer time for refactoring to trade for smaller number of refactoring cycles, and vice versa.
  - Prioritize refactoring tasks to address the critical areas first.
  - Involve product owners and stakeholders to get their perspectives

Methods to Manage Technical Debts:
Discuss Technical Debts

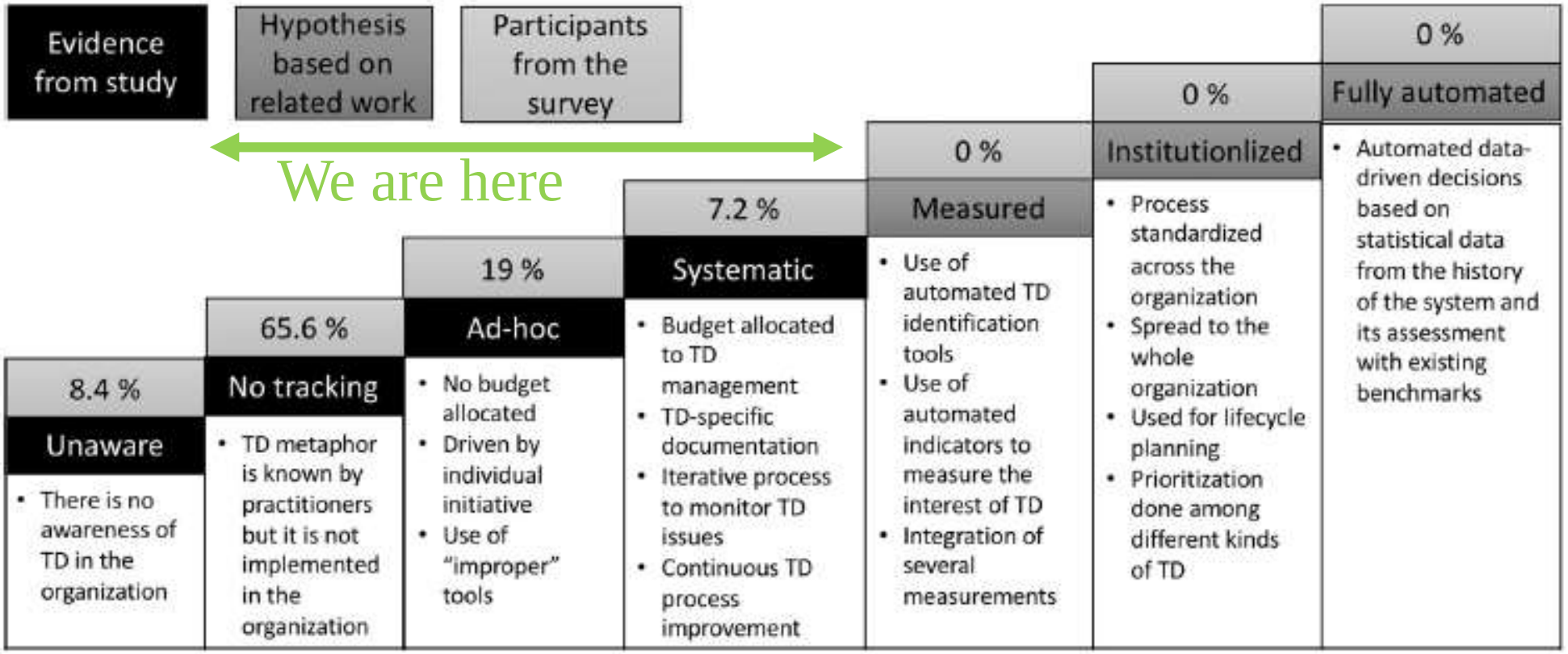
- Helps to educate technical staff about business decision making involved in the project
  - Fit developers into the business context
- Helps to educate business staff about technical decision making
  - Also serve as an aspect of the inputs to their business or operational decisions
- Raises awareness/transparency of important issues that are often covered up in the project
  - As a risk aversion measure
- Allows technical debts to be managed more explicitly and visually
  - As a starting point for a project resolve them one by one.

State of the Practices in Managing Technical Debt

Managing Technical Debts in Practice

- Although most developers may have expressed a desire for better ways of doing all these things, and yet they did not do it.
- Strategies in workplaces (from a survey [5] on 26 companies)
  1. Do nothing—"if it ain't broke, don't fix it"—because the debt might not ever become visible to the customer.
  2. Use a risk management approach to evaluating and prioritizing technical debt's cost and value. E.g., allocate 5 to 10 percent of each release cycle to addressing technical debt.
  3. Manage the expectations of customers and nontechnical stakeholders by making them equal partners and facilitating open dialogue about the debt's implications.
  4. Conduct audits with the development team to make technical debt visible and explicit; track it using a backlog/task board.

Managing Technical Debts in Practice [7]



A survey [7] conducted in 2017: 226 participants from 15 large organizations with different roles (developers, architects, managers)

Managing Technical Debts in Practice [7]

highest

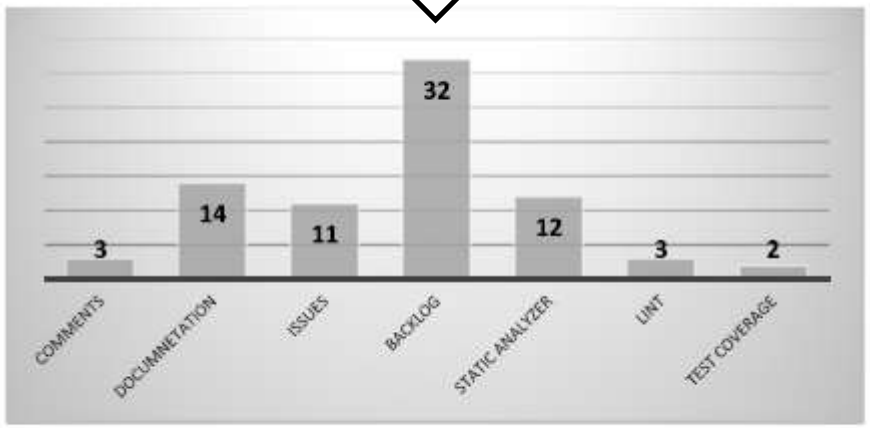


Fig. 13. Number of participants using a specific kind of tool.

lowest

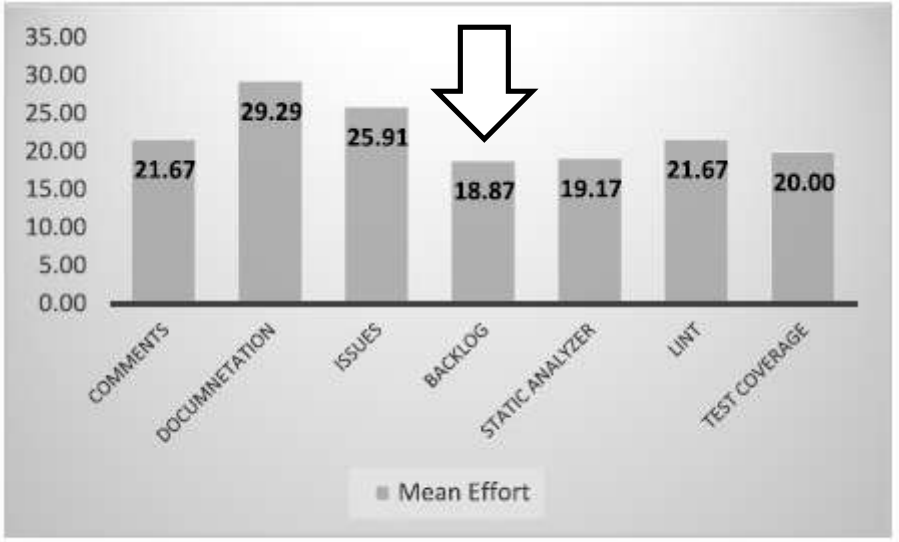


Fig. 15. Mean of management effort for each kind of tool.

See the next slide for the qualitative analysis on these tools

Managing Technical Debts in Practice [7]

- **[Effective] Backlogs, static analyzers and "lint" programs** all **increase the tracking level**, but we cannot see a big difference (although static code analyzers seem to contribute better to the participants' awareness). They are also the ones with the **least overhead**. They therefore seem to be considered the **best practices** at the moment **to track TD**.
  - **[Effective] Backlogs are the most used tool** among the participants. In particular, the most used backlog tools are Jira, Hansoft, and Excel.
- **[Ineffective] Comments** in the code help awareness, but they are **not considered tracking**, and they are used by just 1% of the respondents. This is probably because they are not used in a document that can be monitored by the team outside the code.
- **[Ineffective] Documentation** increases TD awareness, but it is **not considered as a high level of tracking**, and it has the **highest overhead**. The main tools used here were Microsoft Excel or Word. We can infer that this practice is **not recommendable** in comparison with the other ones.
- **[Ineffective]** Using a **bug system** for tracking TD is not considered as contributing to a better level of awareness or tracking compared to the other techniques, and it has a slightly higher overhead. We would infer that this is also **not the best way of tracking TD**.
- **[Ineffective] Test coverage does not** seem to **contribute** too much **to the awareness and tracking level**, although it does not involve much overhead. This might be because test coverage is related to only a small part of TD.

References and Resources

1. Ward Cunningham. 1992. The WyCash portfolio management system. In Addendum to the proceedings on Object-oriented programming systems, languages, and applications (Addendum) (OOPSLA '92), Jerry L. Archibald and Mark C. Wilkes (Eds.). ACM, New York, NY, USA, 29-30. DOI=http://dx.doi.org/10.1145/157709.157715
2. S. McConnell, "Managing technical debt (slides)," in Workshop on Managing Technical Debt (part of ICSE 2013): IEEE, 2013
   - http://2013.icse-conferences.org/documents/publicity/MTD-WS-McConnell-slides.pdf
3. Philippe Kruchten, Robert L. Nord, and Ipek Ozkaya. 2012. Technical Debt: From Metaphor to Theory and Practice. IEEE Softw. 29, 6 (November 2012), 18-21. DOI: https://doi.org/10.1109/MS.2012.167
4. Chris Sterling, 2010. Managing Software Debt: Building for Inevitable Change. Addison-Wesley Professional; 1 edition, ISBN-10: 0321948610. [a good resource to know technical debts]
5. Lim, N. Taksande, and C. Seaman, "A balancing act: What software practitioners have to say about technical debt," IEEE Software. DOI: 10.1109/MS.2012.130
6. Jesper Boeg, Successfully balancing technical debt and new features – staying in the "flow-zone" or how to get back in there. http://agileupgrade.com/successfully-balancing-technical-depth-and-new-features-staying-in-the-flow-zone-or-how-to-get-back-in-there/
7. Antonio Martini, Tersese Besker, Jan Bosch: Technical Debt tracking: Current state of practice: A survey and multiple case study in 15 large organizations. Sci. Comput. Program. 163: 42-61 (2018)
8. Terese Besker, Antonio Martini, and Jan Bosch. 2019. Technical debt triage in backlog management. In Proceedings of the Second International Conference on Technical Debt (TechDebt '19). IEEE Press, Piscataway, NJ, USA, 13-22. DOI: https://doi.org/10.1109/TechDebt.2019.00010
9.