The Technical Debt Metaphor: Principles, Strengths, Limits, and Tool Support

Davide Falessi
University of Rome “Tor Vergata”
January 8, 2014

Fraunhofer USA
Center for Experimental Software Engineering

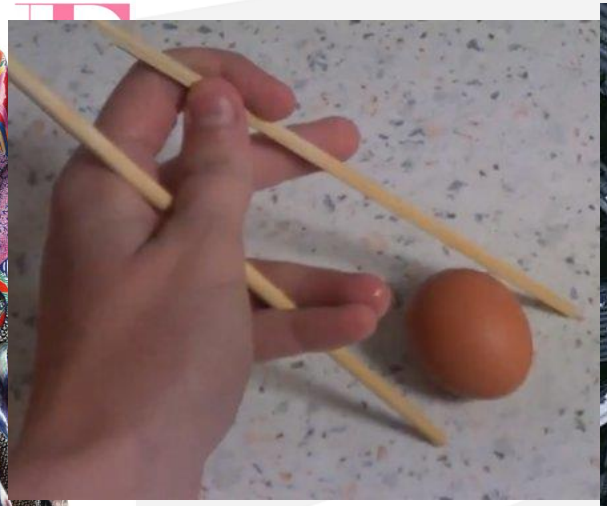
Agenda

- Aim
- Definitions and Vision
- Research directions
- Tool Support
- Requirements of a next generation tool support
- References

Aim

- Provide an overview about Technical Debt

- Why?



Fraunhofer USA
Center for Experimental Software Engineering

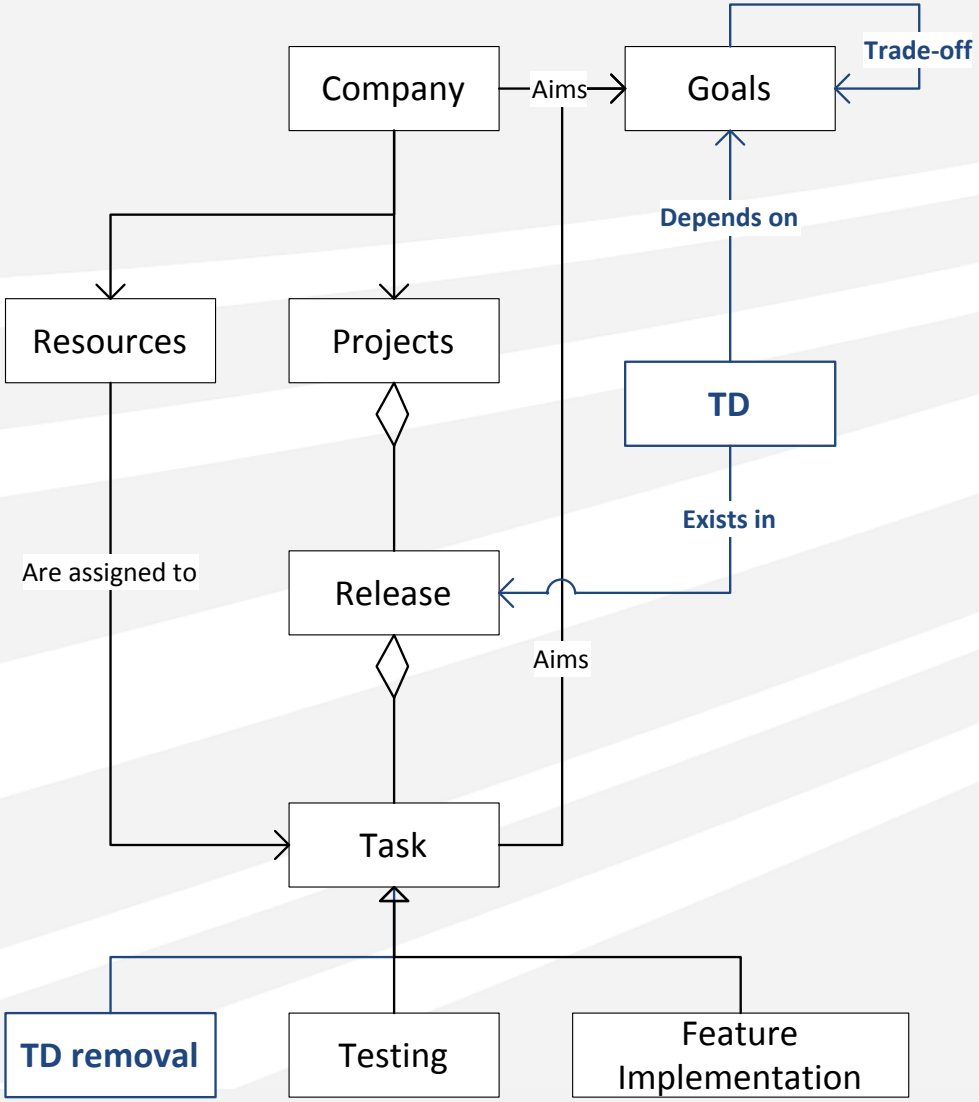
Definitions

- Technical debt is a **metaphor**.
  - Pros: widely applicable.
  - Cons: wrongly used.

Definitions

- TD can be seen as the result of an optimization for the short term which leaded to long term handicaps.
  - Examples: low comments in code, high complexity
- Technical Debt:
  - Can emerge organically as every system. Every system, while evolving, improves complexity.
  - Can be opportunistically chosen (“let’s release now, we’ll deal with it later on”).

My Vision



Fraunhofer USA
Center for Experimental Software Engineering

Definitions

- TD consists of two important concepts:
  - **Principal**: the cost of eliminating the debt.
  - **Interest**: the penalty to be paid in the future if the debt is not eliminated.
    - E.g.: a high complex module could require significant effort to be refactored (principal), however, if not refactored, it could slow down development speed (interest).

TD vs. SW Quality

- TD is **not new**.
- SW quality **includes** TD.
- TD is very close to **Maintainability** concept.
- The concept of technical debt proved to be useful:
  - Large organizations (e.g., Cisco, Siemens, Lockheed Martin, etc.) have explicitly introduced it in some form or another in their software development process, as something to identify, value, and take into consideration while planning iterations and releases.

TD vs. SW Quality

- TD is **not new**.
- SW quality **includes** TD
- TD is very close to **Maintainability** concept.
- The concept of technical debt proved to be useful:
  - Large organizations (e.g., Cisco, Siemens, Lockheed Martin, etc.) have explicitly introduced it in some form or another in their software development process, as something to identify, value, and take into consideration while planning iterations and releases.

Important considerations

- Can **expire**
- **All projects** have some TD
- Not always good to **remove**
- NOT just **defects**
- NOT lack of **process**
- NOT the new **features** not yet implemented
- Depends on the **future**
- Must be **estimated** (Can’t be measured)
- Changes when changing **goals**

Use Cases

- Principal **quantification**.
- Interest **quantification**.
- **Prioritization** of activities.
- Support **Intentional/Explicit trade-off** between short-term vs. long-term goals, e.g. refactoring or coding?
- **Effort allocation** for a healthy project (e.g. 20% testing, 20% removing debt, e.g.)
- **Controlling evolution** of TD during time within a project.
- Suggesting the **set of quality rules to fix** for minimizing TD (given a time-to-market constraint).

Research directions

- Metrics
- Release planning
- Empiricism
- Decision-making
- Estimation

Demo

- Tool support: just a set of quality rules!

- SonarQube (www.sonarqube.org)

6,500+	downloads per month
1,500+	active contributors
50+	plugins in the open source forge
300,000+	downloads
50,000+	instances in the world

- Local vs http://nemo.sonarqube.org/

Requirements of a useful tool

- **R1:** Managing principal, interest, and time-to-market.
- **R2:** Translating decisions into economic consequences
- **R3:** Managing uncertainty in a rigorous way
- **R4:** Managing the evolution of economic consequences
- **R5:** Balancing rigor and ease of use via scalability

Requirements of a useful tool

- **R6:** Completeness and integration
- **R7:** Balancing expert opinions and automated estimates
- **R8:** What-if analysis as interpretation of possible distributions
- **R9:** Sensitivity analysis
- **R10:** Scenario analysis

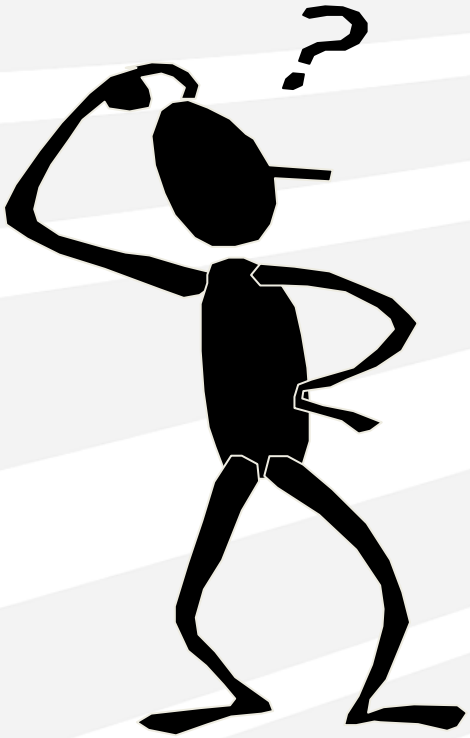
Future steps

- **Principal quantification**
  - A perfect system with no debt is unfeasible
  - Scenarios exist in which the debt of a (good) project exceeds its profit. Wrong message to both developers and managers.
- **Interest** quantification:
  - Interest is not quantified in the same terms as the principal, so it is hard to trade off principal and interest.
  - Interest is not based on historical data.
- More **artifacts** than just code.

References

- P. Kruchten, R. L. Nord, I. Ozkaya, and ***D. Falessi***, “Technical debt: towards a crisper definition report on the 4th international workshop on managing technical debt.” ACM SIGSOFT Software Engineering Notes, Vol. 38, Is. 5 Pages: 51-54.
- ***D. Falessi***, M. Shaw, F. Shull, K. Mullen, M. Stein, “Practical Considerations, Challenges, and Requirements of Tool-Support for Managing Technical Debt”, 4th International Workshop on Managing Technical Debt (MTD 2013), co-located with ICSE, San Francisco, USA, 2013.
- F. Shull, ***D. Falessi***, C. Seaman, M. Diep, L. Layman, “Technical Debt: Showing the Way for Better Transfer of Empirical Results”, Perspectives on the Future of Software Engineering, pp 179-190, 2013.

Contact Information



Davide Falessi

dfalessi@fc-md.umd.edu

240-487-2928

Fraunhofer USA

Center for Experimental Software Engineering