Technical Debt:
At the Intersection of Decades of Empirical Software Engineering Research

Carolyn Seaman
University of Maryland Baltimore County
Fraunhofer Center for Experimental Software Engineering

Workshop on Managing Technical Debt
Baltimore, MD
October 9, 2013
"It is only a little hyperbolic to call this a watershed moment for empirical [software engineering] study, where many areas of progress are coming to a head at the same time."

Contributing Streams of Research

- Software aging and decay
- Risk management
- Qualitative methods and appreciation for context
- Software metrics
- Program analysis
- Software quality
Software Aging and Decay

- Foundational
  - Technical Debt is in some ways just a restatement of these ideas
- Another metaphor
  - Like human aging
  - Changing the software becomes harder as it evolves
- Results
  - Inability to keep up
  - Reduced performance
  - Decreased quality
- Lehman’s Law of Increasing Complexity:
  - Complexity increases unless work is done to maintain or reduce it

Lehman and Belady, 1985
Parnas, 1994
Also foundational
- Instances of Technical Debt constitute one type of software risk
- Risk Management cycle (identify, assess, manage) provides a template for managing Technical Debt
- Risk Assessment approaches (e.g., Risk exposure analysis) provides ways to quantifying Technical Debt
- Concept of utility loss provides a way to characterize the interest on Technical Debt
The Evolution of Qualitative Methods in SWE

Qualitative methods non-existent in software engineering research

First published studies

Era of political correctness - special interest

Qualitative methods accepted and widespread

1998
2000’s
2013

Seaman, 1998
Dyba et al., 2011
Dittrich et al., 2007
Empirical software engineering researchers can now add a host of qualitative methods to their empirical toolkit.

Many good examples of qualitative studies are available in the literature (e.g. in special issues).

Many experts who are highly experienced.

Starting to look at qualitative synthesis of studies (e.g. in the context of SLRs).

**Bottom line:** We now have the tools and expertise available to fully investigate questions of human behavior and context.
Technical Debt related concepts are context-specific

A project’s Technical Debt strategy should be based on goals and “pain points”

Context factors can be elicited in a number of ways

We need qualitative methods to ensure capture of all relevant factors

Qualitative work in Technical Debt research one of the reasons for its relevance to practice

**Bottom line:** We can’t study TD properly without qualitative methods, and until recently we didn’t as a community know how to use qualitative methods effectively
The Evolution of Software Metrics

- Early complexity metrics
- Institution of metrics programs
- Visualization and dashboards
- Modern development environments and tools

1970’s 1990’s 2000’s 2010’s

McCabe, 1976
Halstead, 1970
Basili et al., 1994
Gaudin, 2009
Schumacher et al., 2010
Bohnet and Döllner, 2011
Snipes et al., 2011
Adoption of software metrics in industry is still spotty
   - Especially in small and medium organizations
   - Many large development organizations are “data-rich” environments
   - Metrics no longer have to be “added on” at the end of the process – better integration is possible
   - **Bottom line:** Tools are available to integrate data collection, analysis, and visualization into the software development process
The relationship between software metrics and Technical Debt is complex and subject to further research.

Not evident that modules with “worse” indicators have “real” debt.

Code smell definitions try to get at the complicated relationship.

In practice, TD management often begins with monitoring metrics.

**Bottom line:** Simple views of metrics are not sufficient; we need easy ways to combine and visualize custom-fit combinations and relationships between different metrics.
The Evolution of Program Analysis

Control and data flow analysis
Principles of OO design
Automation of anomaly detection; code smells
Modern development environments and tools

1970’s | 1980’s | 2000’s | 2010’s

Kildall, 1973 | Ball and Rajamani, 2002
Jones, 1981 | Munro, 2005
Rentsch, 1982 | Bohnet and Döllner, 2011
Booch, 1986 | Snipes et al., 2011
Current State of Program Analysis

- A plethora of tools available
- Easy to use
- Some cases in which program analysis is integrated into the build process
  - Even quantitative thresholds for an acceptable number of “issues”
- Generate mountains of information
- **Bottom line:** the challenge is to make sense of the analysis results – what’s important?
Like software metrics, the relationship between program analysis and Technical Debt is complex.

Anomaly detection through program analysis (e.g. code smells, ASA “warnings”, etc.)
- Not clear what anomalies constitute debt
- Tools don’t usually convey information about the value or importance of the anomaly

Program analysis provides the building blocks for techniques that look at higher-level (e.g. architectural) issues.

**Bottom line:** Modern program analysis techniques provide tools only facilitate identification of debt, they need more support to identify and evaluate instances of “real” debt.
The Evolution of Software Quality

Quality equals no defects

Recognition of the "ilities"

Value-based software engineering

Quality is contextual

1970’s

2000’s

Rubey & Hartwick, 1968
Boehm, 1973
Biffl et al., 2005
Current State of Software Quality

- Organizations still struggle to define quality in a meaningful way
- Maturity of understanding of quality varies
- Organizations who manage quality successfully have tied their quality indicators to business goals and desired outcomes
- **Bottom line**: Quality management is goal-driven
Technical Debt is primarily concerned with the maintainability aspect of quality.

But most other “ilities” feed into maintainability.

The debt-related concepts of principal and interest are directly tied to the idea of value.

The idea that quality contributes to value, not just function.

**Bottom line:** We now understand that quality means different things in different times and places, and it is this understanding that is crucial for the study of Technical Debt.
Persistent Problems in Empirical Software Engineering

Technology Transfer

Evolution of the Discipline
Too few empirical software engineering researchers get to see their ideas put into practice

Our research too often does not start from a real problem or a real context

Our research too often is described in terms that are not relevant for practitioners

We’re not good salespeople

Requirements of publication and practice are not always in harmony
Technology Transfer and Technical Debt

The Technical Debt metaphor
- Gives us a vocabulary that both researchers and practitioners understand
- Is a problem that practitioners care about
- Forces researchers to view the problem from a practice point of view

Applying Technical Debt research in practice starts with identifying the project’s sources of “pain”

Thus, research in this area by necessity is grounded in practice
Software engineering research has long suffered from an inability to build on previous results. Too often suffers from a lack of grounding in prior literature. Previous slides show successes in individual areas. But we need to get better at applying findings in one area to solve problems in another, combining diverse solutions to address a multi-faceted problem, and see the relationships between different areas.
Technical Debt is a multi-faceted problem
Addressing it effectively in practice relies on solutions from:
- Software evolution
- Risk Management
- Qualitative assessment of context
- Software metrics
- Program analysis
- Software quality

Here’s our chance to appreciate and use results from outside our own corners of the field
Recap

Technical Debt

- Is a metaphor that describes a real problem in software engineering practice
- Requires solutions from a variety of different areas in empirical software engineering that have evolved over the last few decades
- Requires solutions that are only now possible because of the level of evolution of these contributing areas
- Provides the potential for addressing some long-term problems in the empirical software engineering research community
Call to Action

- Do more research on Technical Debt, **BUT**
  - Don’t lose the industry focus
    - Keep talking to practitioners
    - Learn the vocabulary
    - Listen to where the pain is
  - Don’t reinvent the wheel
    - Read the literature
    - Adapt solutions
    - Collaborate
“Watershed”? 

“It is only a little hyperbolic to call this a watershed moment for empirical [software engineering] study, where many areas of progress are coming to a head at the same time.”

- Are we at a historical moment in empirical software engineering research?
- Will everything be fundamentally different from now on?
- We have the right problem, we have a history of research providing at least the beginnings of the right solutions.
- It could be....
Thank you!

Questions?
References I


References II