Quantify design quality. It is the crux of instrumenting a lean transformation.

Business agility is as much a function of architecture as it is of process.

Are you wasting too much time in overhead work?
Everyone involved in a software supply chain is perturbed by this provocative statement:

> *40 to 60% of your resources are consumed in non-value-added effort*.

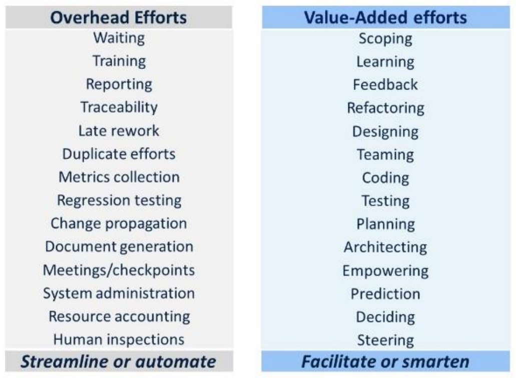
Executives, project leaders, and developers know they spend a huge amount of time and effort in waste, rework, and other unnecessary overhead activities that need to be streamlined, automated, or eliminated. Everyone feels this pain.

What is overhead work? What is value-added work?
This topic rattles nerves. A precise answer must be context-dependent, but the following definitions are based on categorizing the artifacts produced in software delivery, where increments of usable functionality are produced by constructing a deliverable product and supplemental artifacts.

- **Value-added work** involves creating the deliverable product. This is what users buy. It is captured in the primary intellectual capital: designs, code, data, tests, and build scripts.
- **Overhead work** involves creating supplemental process artifacts like plans, requirements, models, progress reports, quality assessments, traceability, training, and proof of compliance .

Some overhead is necessary to manage a software delivery. However, when organizations become inefficient and fatty, it is usually because of unnecessary overhead work.

With a basic understanding of what is value-added and what is not, your teams can better reason about where to improve efficiency. This is the crux of lean transformation and improved software economics: Minimize the resources consumed on non-value-added work, freeing up more resources for value-added work. When a team explicitly differentiates overhead activities from productive activities, as shown in Figure 1, discussions heat up. Try this exercise with teams in your own context.



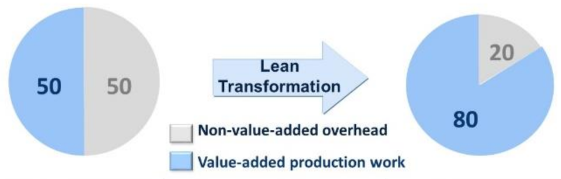
Overhead Efforts	Value-Added efforts
Waiting	Scoping
Training	Learning
Reporting	Feedback
Traceability	Refactoring
Late rework	Designing
Duplicate efforts	Teaming
Metrics collection	Coding
Regression testing	Testing
Change propagation	Planning
Document generation	Architecting
Meetings/checkpoints	Empowering
System administration	Prediction
Resource accounting	Deciding
Human inspections	Steering
Streamline or automate	***Facilitate or smarten***

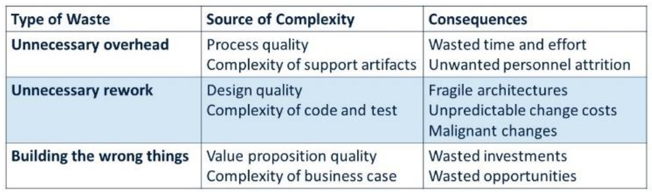
Figure 1: A typical allocation of overhead vs. value-added work

Start by dividing your activities into two lists: overhead activities and value-added work. This exercise alone is a powerful catalyst for debating what is value-added and what is not. Then prioritize the lists to identify the top few overhead efforts that are consuming too many resources and the top few value-added efforts that would benefit most from more resources. Whether you manage engineers, marketers, operators, developers, or finance professionals, this process will result in eye-opening debates.

Two recurring themes have surfaced from such exercises. First, the top item on the overhead list is invariably late rework. The primary root causes of late rework are poor design quality and protracted design verification. Second, the most wasteful overhead activities are usually burdened onto developers, and the value-added improvement priorities are targeted at the leadership team. In most organizations, the bottleneck is at the top of the bottle.

Where does all this wasted effort come from?
Figure 2 illustrates three primary sources of waste: unnecessary overhead, unnecessary rework, and building the wrong things.





Type of Waste	Source of Complexity	Consequences
Unnecessary overhead	Process quality Complexity of support artifacts	Wasted time and effort Unwanted personnel attrition
Unnecessary rework	Design quality Complexity of code and test	Fragile architectures Unpredictable change costs Malignant changes
Building the wrong things	Value proposition quality Complexity of business case	Wasted investments Wasted opportunities

Figure 2: Complexity translates into waste, rework, and overhead

In a perfect world, developers would understand macro-level design intentions, code the individual components, integrate them, and deliver an error-free system into production. In our imperfect world, we must introduce supplemental artifacts to manage teams of error-prone humans, who communicate ambiguously, and we must rework most artifacts multiple times to deliver complex software system. It is this complexity, and the resulting miscommunications and human error, that result in higher overhead. Complexity of architecture, complexity of code, complexity of communications, complexity of process, and complexity of change all contribute to overhead.

Software is both complex and complicated. It poses significant challenges for teams of people to understand and communicate in unambiguous ways. For executives, architects, and project managers, this complexity translates directly into uncertainty and high variability in outcomes.

How do we reduce uncertainty and better manage complexity?
One starting point is better measurement.

> *Scientists define measurement as an observation that reduces uncertainty, where the result is expressed as a quantity.*
> (Douglas Hubbard, *How to Measure Anything,* 2010)

The foundations of agile methods, DevOps principles, and lean systems engineering revolve around measuring velocity, using smaller batch sizes, failing fast, improving collaboration, accelerating feedback cycles, and reducing waste. These techniques manage complexity implicitly and explicitly by quantifying progress and quality trends to attack uncertainty earlier.

Stephen Covey coined the term “the speed of trust” to identify trust as the element necessary to reduce overhead and improve efficiency and effectiveness (*The Speed of Trust*, 2008). The speed of trust can also be appreciated by its logical opposite: the slowness of distrust. In most software enterprises, overhead activities are proportional to the amount of distrust. Complexity leads to uncertainty, which leads to distrust. Some distrust is healthy, because humans make errors and poor judgments. Some levels of oversight, planning, reporting, documentation, and assurance are needed to deliver quality outcomes predictably. However, when overhead activities waste resources, the value achieved is out of balance with the cost and inconvenience. The system becomes inefficient and outcomes degrade.

How can we measure design quality?
Quantifying design quality is one of the software industry’s holy grails. Our research and field applications demonstrate valuable insights that can be realized through code scans and visualized through design structure matrices (DSMs). These are a few axioms of design quality:

1. Design quality is the dominant factor in long-term software economic outcomes.
2. Better design quality is analogous to a lower interest rate on technical debt.
3. Design quality drives the breadth and depth of interpersonal communications across the enterprise.
4. Design quality involves economic tradeoffs between efficiency (resources consumed) and effectiveness (user-delivered value).
5. Design quality is best measured by quantifying structural analytics (efficiency) and integrated test analytics (effectiveness).

The last axiom asserts two specific classes of measurement. *Structural analytics* provide quantifiable measures of design quality that can be extracted from a code base. More structural complexity leads directly to more overhead and to inefficiencies in managing communications among people and activities across teams. *Integrated test analytics* provide measures of defects and change trends from configuration control and issue tracking tools. Quantifying the resources consumed to change a system allows teams to understand the consequences of structural complexity and design quality in economic terms.

The word *agility* means speed of change. Your change speed must be an asset, not an anchor. Correlating change costs for defects, new features, and other engineering changes allows validated learning to be optimized and projects to be steered toward better economic outcomes. *Software agility is as much a function of architecture as it is of process.* The most important characteristic of software is that it is “soft.” The easier software is to change, the easier it is to achieve any of its other needed attributes.

How can we start transforming?
Efficiency in execution is best achieved through improved design quality and reduced complexity. Software delivery teams should target 20 to 30% overhead. Quantifiable improvements in efficiency are usually step 1 in a lean transformation, as shown in Figure 3. Cost analytics are more mature than value analytics. Nothing raises the morale of developers more than reducing overhead.

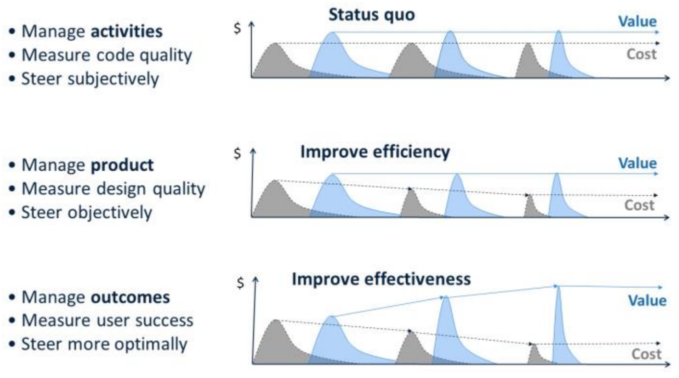


Figure 3: Improve lean efficiency, then improve effectiveness

We recommend using these four primary threads to transform the efficiency and effectiveness of software delivery incrementally.

1. **Steer**: Measure the product artifacts, *not* the process artifacts, for more honest insight. Measuring design quality enables optimized steering and reduces the overhead and waste of late rework.
2. **Develop:** Accelerate feedback cycles through agile methods and early design quality verification. Agile methods shift emphasis from the overhead of supplemental artifacts to the value-added artifacts of design, code, and test.
3. **Deploy:** Automate the build and release process to reduce friction during deployment. Checkpointing design quality and complexity growth with each deployment makes technical debt manageable.
4. **Collaborate:** Unify methods and tooling across the software supply chain for holistic efficiencies. Measuring code quality and design quality balances user-perceived quality with economic improvements.

Silverthread’s capability and know-how can be a catalyst for your lean transformation. Quantifying design trends encourages more honest and trustworthy exchanges among stakeholders. Increasing trust enables leaner production by reducing sources of overhead, unnecessary rework, and waste. Trust is the currency of lean engineering efficiency.

Contact Us

Silverthread’s mission is to advance the state of software measurement practice by quantifying complexity and design quality. Our measurement know-how can establish a more trustworthy foundation for improving software economics.
http://silverthreadinc.com