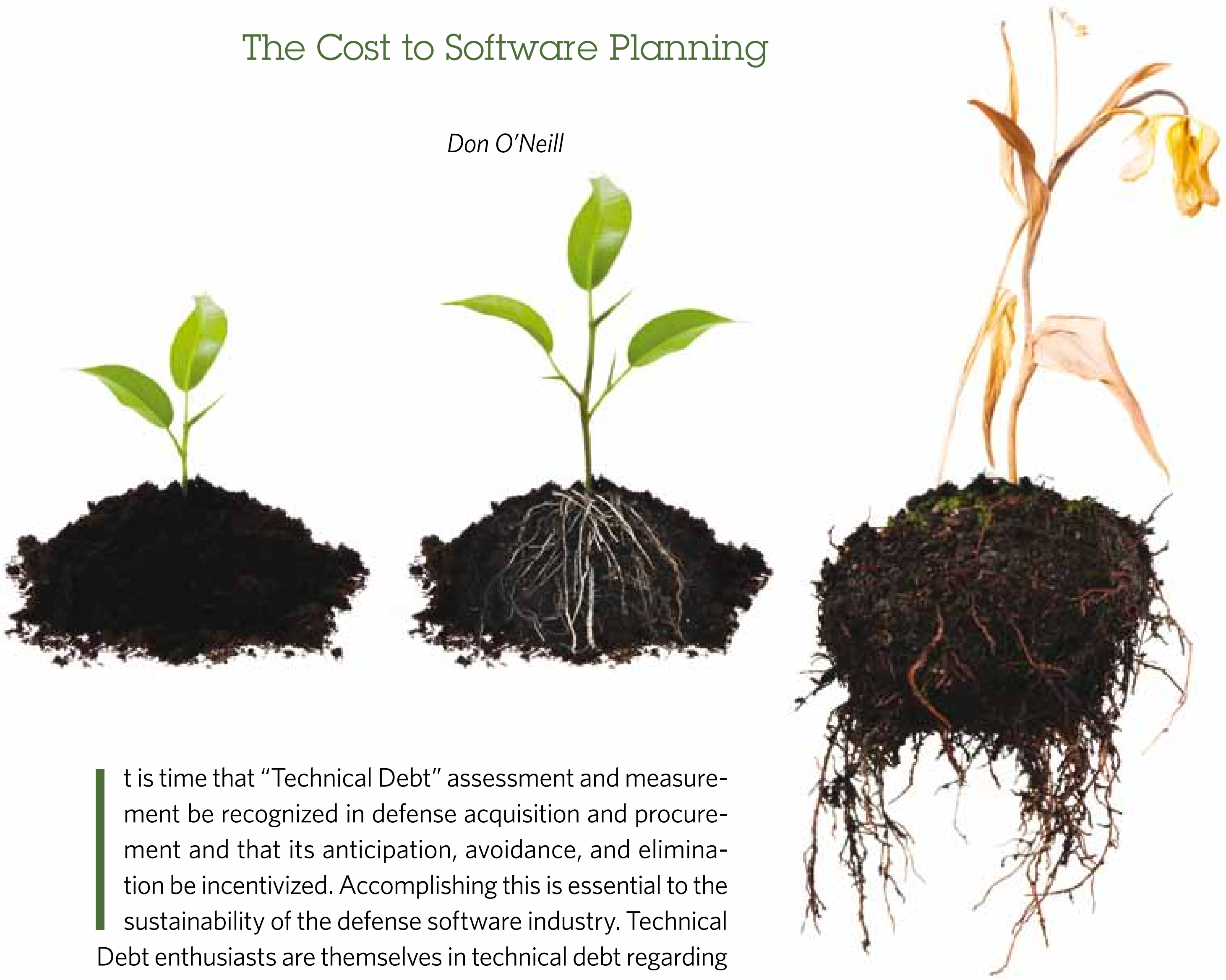
‘Technical Debt’ in the Code

The Cost to Software Planning

Don O’Neill



It is time that “Technical Debt” assessment and measurement be recognized in defense acquisition and procurement and that its anticipation, avoidance, and elimination be incentivized. Accomplishing this is essential to the sustainability of the defense software industry. Technical Debt enthusiasts are themselves in technical debt regarding its definition. It is time to put a finer edge on this definition and update it. The early, archaic, and somewhat awkward definition, introduced by Ward Cunningham in 1992, is, “Not quite right code which we postpone making right.”

Instead, I suggest the following definition: Technical Debt is the organizational, project, or engineering neglect of known good practice that can result in persistent public, user, customer, staff, reputation, or financial cost.

Scope of Technical Debt

The current scope of Technical Debt as a metaphor for the consequences of neglect in software engineering and management is somewhat old-style and certainly programmer-centric. This scope of Technical Debt from the viewpoint of the programmer is one of software components, code and test activities, and static analysis. However, the neglect for which the project and enterprise will pay in terms of interest on the debt includes systems and software

O’Neill *was president of the Center for National Software Studies in 2005 to 2008. Following 27 years with IBM’s Federal Systems Division, he completed a 3-year residency at Carnegie Mellon University’s Software Engineering Institute (SEI) under IBM’s Technical Academic Career Program and has served as an SEI Visiting Scientist. He is a mathematician, a seasoned software engineering manager, and an independent consultant.*

engineering and management, systems and systems of systems, iterative life cycle model dynamics, dynamic analysis, and finite word effects. So, clearly, the scope of Technical Debt must be elevated.

Technical Debt is an interesting metaphor. Its utility lies in its simplicity and ease with which complex software planning and technical issues can be framed for executives and managers who may lack the technical background to engage these issues firsthand. This shorthand method of framing complex problems leads to loss of underlying detail that can restrict or misdirect the identification, analysis, and resolution of software planning and technical issues among those who do possess the technical background to engage these issues firsthand.

For starters, Technical Debt involves more than the technical and engineering dimension; it also involves software engineering process and management.

The success of large-scale software intensive systems largely depends on the engineering, management, and process capabilities, people, practices, methods, and tools of the enterprise charged with the requirements determination, design, development, testing, fielding, and sustainment of systems and systems of systems. Within any organization, these elements of success are in various stages of maturity, and their evolution and alignment may become the source of strategic software management and continuous process improvement. At any time, these gaps can be referred to as Technical Debt when they result in persistent costs and risks to reputation, economics, mission, or competitiveness.

When these gaps are neglected, whether undiscovered or consciously ignored, Technical Debt may be incurred.

Technical Debt, then, is the organizational, project, or engineering neglect of known good practice that can result in persistent public, user, customer, staff, reputation, or financial cost. Shortcuts, expedient activities, and poor practice contributing to the initial product launch or initial operational capability often are cited as justifiable excuses in taking on Technical Debt. But, in truth, most Technical Debt is taken on without this strategic intent, without even knowing it, and without the wherewithal in capability or capacity to do the job right.

In any event, as the twig is bent so grows the tree, and the weight of accumulated Technical Debt immediately and continuously extracts its cost on the organization.

Sources of Technical Debt

Technical Debt is considered written off only when it is eliminated. Draining the swamp depends on understanding and aligning the sources of Technical Debt in management, engineering, and process.

Sources of Technical Debt in engineering involve neglect in application domain understanding, requirements determination, system and software architecture, iterative multilevel design, staged incremental development, software development life cycle, programming language, middleware, operating system, network interface, and software development environment.

Sources of Technical Debt in management involve neglect in requirements management, estimating, planning, measurement, monitoring and controlling, risk management, process management, team innovation management, supply chain management, team building, personnel management, and customer relationship management.

Sources of Technical Debt in process involve insufficient evidence of explicit goals and readiness to perform, insufficient accountability based on work responsibility matrix, insufficient planning of design levels and staged increments, and insufficient planning, management, and control of software product releases.

Some argue that Technical Debt should be limited to intentionally deferred work as though incurring Technical Debt is a calculated risk. However, in the heat of battle on a project looking for shortcuts to meet cost and schedule, there is no calculation. There is only expediency.

Suppose there was a calculation. What would it look like?

Would it accord a cost benefit for rework of deferred effort? No, doing it right the first time is more cost-effective. Doing it later, perhaps with less skilled personnel, may mean doing it yet again and again.

Would it accord a cost benefit if the rework of deferred effort was never needed at all? Yes, uncertainties that a calculated risk might consider include banking on the possibility that the initial effort will become a throwaway prototype or that the demand for modernization will overtake the project before the rework of deferred effort is performed.

Would it accord a schedule benefit if function were postponed or some functionally equivalent shortcut were adopted for the moment? Yes, doing less work should take less time.

Would it accord a schedule benefit if "going fast" entails abandoning the organization's standard of excellence in disciplined software engineering and drifting into a stream of consciousness, ad hoc hacking style of programming? No, the ad hoc programming style will result in a higher defect rate that will impact testing and fielding. Ad hoc programming does not deliver superior results with respect to cost, schedule, quality, and performance. Delivering on these attributes takes engineering. So when thinking about "going fast," it may actually pay to go slow.

For those who reserve Technical Debt for intentional deferment of effort, there may be a sort of pride in going against the grain of good, disciplined software engineering practice—as if their superior skills will permit them to dodge a bullet of some kind during development, only to patch up the situation later when the coast is clear. In my experience, the coast rarely is clear, the rework is ignored or gets done later by less skillful people, and the interest paid and higher cost to fix later defeat competitiveness.

Another category of Technical Debt is not intentional and centers around the "neglect of known good practice." Perhaps some feel neglect is too harsh a term; perhaps others reserve Technical Debt for intentional deferment of effort. In either case, the result is deferred work with consequences that extract an ongoing cost and the postponed elimination of which will cost more than doing it right the first time.

Technical Debt from all sources needs to be on the table when the full cost of rework is weighed against the cost of additional functionality or the cost of a modernization program.

Technical Debt, Triggers, and Analytics

Technical Debt is the organizational, project, or engineering neglect of known good practice that can result in persistent public, user, customer, staff, reputation, or financial cost. When adopted, Technical Debt becomes the hole in your canoe. Each gallon of water bailed incurs additional cost. Each gallon of water not bailed adds to the sluggishness of the operation.

Technical Debt refers to postponed or deferred work, whether by intent or by neglect. Incomplete or shoddy work extracts a persistent cost on ongoing software operations. In addition, corrective rework costs more than doing it right the first time. Technical Debt typically is viewed as a problem to recover from once it has occurred. However, a better strategy is systematically to anticipate and avoid the conditions that contribute to Technical Debt in the first place.

The methods to anticipate systematically and avoid Technical Debt need to be built into the software development life cycle. The intended outcomes include on-budget, on-schedule deliveries of defect-free components and systems traceable to requirements with managed and controlled frequency of releases that sustain user operations. Project assessment focuses on the cost, schedule, quality, and performance triggers that serve as the preconditions for Technical Debt.

Technical Debt is considered written off only when it is eliminated at the source, including management, engineering, and process. What are the conditioning triggers for each source?

Sources of Technical Debt in management involve neglect in requirements management, estimating, planning, measuring, monitoring and controlling, risk management, process management, team innovation management, supply chain management, team building, personnel management, and customer relationship management.

The Technical Debt conditioning triggers for management are shown in Table 1.

Sources of Technical Debt in engineering involve neglect in application domain understanding, requirements determination, system and software architecture, iterative multilevel design,

Table 1. Technical Debt: Management, Trigger, Condition, Action

Source	Trigger	Condition	Action
Management	M1. Prioritized goals	Where schedule or cost is accorded priority over defect free delivery	A Technical Debt conditioning trigger is set.
	M2. Organization levels	Where the software function is separated from program management by two or more levels	A Technical Debt conditioning trigger is set.
	M3. Schedule	Where the number of months planned is less than the estimated month at completion	A Technical Debt conditioning trigger is set.
	M4. Cost	Where the budget at completion is less than the estimate at completion	A Technical Debt conditioning trigger is set.
	M5. Milestone completion	Where the completion schedule for any milestone completion planned date is replaced with a replanned date	A Technical Debt conditioning trigger is set.
	M6. Headcount and effort	Where overtime, off-the-clock time, and personnel turnover rate is trending upward	A Technical Debt conditioning trigger is set.
	M7. Frequency of release	Where the frequency of release is daily or weekly	A Technical Debt conditioning trigger is set.
Total			Multiple Technical Debt conditioning triggers are set.

staged incremental development, software development life cycle, programming language, middleware, operating system, network interface, and software development environment.

The Technical Debt conditioning triggers for engineering are shown in Table 2.

Sources of Technical Debt in process involve insufficient evidence of explicit goals and readiness to perform, insufficient accountability based on work responsibility matrix, insufficient planning of design levels and staged increments, and insufficient planning, management, and control of software product releases.

The Technical Debt conditioning triggers for process are shown in Table 3.

The author can be contacted at **oneilldon@aol.com**.

Table 2. Technical Debt: Engineering, Trigger, Condition, Action

Source	Trigger	Condition	Action
Engineering	E1. Deep domain expertise	Where deep domain expertise is not widespread on the project	A Technical Debt conditioning trigger is set.
	E2. Software architecture	Where software architecture is not tightly coupled with middleware, operating system, and network services	A Technical Debt conditioning trigger is set.
	E3. Requirements known	Where requirements are not fully known	A Technical Debt conditioning trigger is set.
	E4. Technical risk	Where the source of technical uncertainty in function, form, or fit is high	A Technical Debt conditioning trigger is set.
	E5. Product size	Where product size estimates at completion exceed product size estimates planned	A Technical Debt conditioning trigger is set.
	E6. Complexity	Where cyclomatic or essential complexity trend upward from one product release to another	A Technical Debt conditioning trigger is set.
Total			Multiple Technical Debt conditioning triggers are set.

Table 3. Technical Debt: Process, Trigger, Condition, Action

Source	Trigger	Condition	Action
Process	P1. Software Project Management	Where the software project management mode is low	A Technical Debt conditioning trigger is set.
	P2. Software Product Engineering	Where the software product engineering mode is ad hoc	A Technical Debt conditioning trigger is set.
	P3. Iterative development	Where incremental or iterative development of design levels and delivery stages is not used	A Technical Debt conditioning trigger is set.
	P4. Best practices	Where the use of best practices is rated low	A Technical Debt conditioning trigger is set.
	P5. Metrics	Where metrics are not used	A Technical Debt conditioning trigger is set.
	P6. Quality Assurance	Where quality assurance is not in place and functioning	A Technical Debt conditioning trigger is set.
	P7. Defect rate	Where the actual defect rate including both defect detection and defect correction exceed the expected	A Technical Debt conditioning trigger is set.
Total			Multiple Technical Debt conditioning triggers are set.