Abstract—Much of the emphasis in identifying a software maintenance project’s technical debt involves analysis of the code to be maintained. Our recent analysis of the sources of software maintenance cost have identified the major sources of technical debt for non-developer maintenance organizations to originate from non-technical acquisition and development decisions and practices. As a way of identifying and avoiding these sources, we have developed an initial counterpart of the Technical Readiness Level framework, called the Software Maintenance Readiness Level (SMRL) framework. This paper summarizes the main non-technical sources of software maintenance technical debt, their root causes, a first-cut SMRL framework, and our early experiences in evaluating it.

Keywords—Software life cycle; Life cycle costs; Software maintainability; Software evolution; Software technical debt, Software maintainability readiness level

I. INTRODUCTION

Several current and future trends present challenges involving further escalation of software maintenance costs. These include increasingly rapid changes in technology, competition, and human-computer interaction; increasing interdependencies among systems (e.g., Internets of Things); increasing sources of cyber-physical-human system vulnerabilities; and proliferation of candidate capabilities (e.g., millions of apps). While some life cycle savings could be achieved by making the maintenance organization more efficient, some recent analyses of the sources of software maintenance cost have found that a primary cause of the high costs and difficulties of software-intensive systems’ maintenance and evolution is that the developers are unable, unwilling, unaware, under-resourced, and/or unappreciative of the need to apply foresight in preparing the system to evolve. Much of the added cost and difficulty emerges as technical debt, but the underlying causes are primarily shortfalls in processes and management. Section 2 summarizes the major foresight shortfalls. Section 3 summarizes the resulting major shortfall categories: Life Cycle Management; Maintenance Personnel; and Maintenance Methods, Processes and Tools (MPTs). Section 4 presents the resulting Software Maintainability Readiness Level (SMRL) assessment framework. Section 5 summarizes our early results in evaluating the SMRL framework, and Section 6 presents conclusions to date.

II. SOFTWARE PROCESS FORESIGHT SHORTFALLS

Here are some of the primary sources of software process foresight shortfalls causing significant levels of technical debt.

1) Separate organizations and budgets for software acquisition and maintenance. The acquisition organization will tend to over-optimize on acquisition cost-effectiveness, leaving the maintenance organization unprepared for cost-effective maintenance.

2) Overconcern with the Voice of the Customer, as in Quality Function Deployment.[1] Delighting customers with attractive features often leads to commitments to incompatible and hard-to-maintain capabilities, which could be detected by listening to the Voice of the Maintainer.

3) The Conspiracy of Optimism. The project sponsors are competing for resources with sponsors of other projects, and will tend to be optimistic about what the project will deliver and how much it will cost. They will often try to get well by outsourcing the development to the lowest-cost, technically acceptable bidder. Contractors bidding to perform the project will also tend to be optimistic about what the project will deliver and how much it will cost.

4) Inadequate system engineering resources. The first organization to be impacted by inadequate budgets and schedules will be system engineering. The result will be exponentially-large amounts of technical debt due to poorly-defined interfaces, unaddressed rainy-day use cases and risks, and premature commitments to hopefully-compatible but actually-incompatible COTS products, cloud services, open-source capabilities, and hopefully-reusable components [2]. The inadequate resources provide no opportunity to develop and review evidence of the feasibility (scalability, compatibility, performance, dependability, etc.) of the commitments.

5) Hasty contracting that focuses on fixed operational requirements. If budgets and schedules are tight, and the contract does not require delivery of test and debugging support, architectural descriptions, development support and configuration management capabilities, and latest-release COTS products, these will not be made available.
to the maintainers. Even if contracted-for, these may be dropped or minimized as lower-priority needs if budgets and schedules are tight. Having a fixed-requirements contract is a source of significant delays, particularly as the pace of change in software-intensive systems continues to accelerate.

6) CAIV-limited system requirements. Often, customers’ desired capabilities exceed the available conspiracy-of-optimism budgets, and a Cost As Independent Variable (CAIV) exercise is performed to prioritize the capabilities, and to include only the above-the-line capabilities in the Request for Proposals or Statement of Work. This throws away valuable information on the likely sources of future maintenance activity.

7) Brittle, point-solution architectures. The lowest-cost, technically-acceptable winning bidder will generally commit to a brittle, point-solution architecture addressing only the capabilities in the Statement of Work, minimizing development costs but again escalating maintenance costs.

8) The Vicious Circle. Even when acquisition organizations wish to include the Voice of the Maintainer and invite them to participate in defining a new system, they will often be met with apologies that the maintainers are too busy compensating for the maintainability shortfalls in their current systems to be able to participate.

9) Stovepipe systems. Having different organizations implement increasingly-interdependent systems leads to numerous clashes in coordinating changes across systems with incompatible internal assumptions, infrastructure commitments, user interfaces, and data structures. Regional and national healthcare systems are just one of many examples.

10) Over-extreme forms of agile development, such as rejecting architectural descriptions as Big Design Up Front (BDUF) and saying You Aren’t Going to Need It (YAGNI). This may be true on small projects where the developers continue into maintenance, but will be disastrous if provided to a different maintenance organization, or when the small project grows into a 50-person team coping with evolving a 500K source lines of code (KSLOC) project [3].

III. MAJOR SHORTFALL CATEGORIES

Several classes of organizations generally do not have serious problems with high maintenance cost for their more diverse and dynamic software-intensive systems. Some have developers who continue with the project through its life cycle. Some whose business or mission depends critically on high levels of service employ and support highly-capable in-house maintenance organizations. Classes of organizations most needing to reduce high maintenance costs for their more diverse and dynamic software-intensive systems are those in which Research and Development (R&D) and Operations and Maintenance (O&M) are separately funded and managed; organizations which outsource software maintenance to external companies; and organizations with in-house software maintenance centers that receive and maintain software developed either elsewhere in the organization or externally. For such organizations, three of the primary symptoms of high maintenance costs are (1) life cycle management shortfalls; (2) maintenance personnel shortfalls; and (3) maintenance methods, processes and tools (MPT) shortfalls.

A. Life Cycle Management Shortfalls

Enterprises with separate organizations and budgets for software acquisition and maintenance will initially focus on getting the acquisition right, such as overconcern with the Voice of the Customer. Since the sponsors will be competing with other proposed projects, and the candidate developers competing with other bidders, they will often fall victims to a conspiracy of optimism that leaves them with insufficient budgets for developing the promised capabilities. As above, the first project function to be underbudgeted will be system engineering, leading to inadequate system engineering abilities. Since the promised schedules are also optimistic, developer source selection will lead to hasty contracting that focuses on fixed operational requirements, and often neglects delivery of maintainer needs and full interface definitions among developers. The statements of work will often have CAIV-limited system requirements, leaving little information about sources of subsequent system evolution to architect for. Anyway, the winning developers will minimize development cost by specifying brittle, point-solution architectures. Frequently, the maintainers are belatedly notified that the software will be theirs to maintain. They will have been busy on other projects, with minimal opportunity to understand or influence the systems development. They will have minimal budget to prepare for the system’s maintenance. As a result, their current domain expertise, skills, and support environment may be a major mismatch to the maintenance needs of the new system. Often, the developed software will have numerous hasty patches that were needed to pass its acceptance tests, a good many of which will have undesirable side effects. There is likely to be a large backlog of change requests and changed interfaces with interoperating co-dependent external systems. The development contract may have neglected to specify that the system be delivered with the latest release of the supporting COTS products. In gathering data for the COCOTS cost model, we encountered one project with 120 COTS products, 55 of which were using versions that had become unsupported by the vendor: a major source of technical debt.

B. Maintenance Personnel Shortfalls

Although initiatives such as Acquisition Logistics and Performance-Based Logistics [4][5] often work well for hardware-intensive systems, they tend to work less well for software-intensive systems due to the differences in hardware and software maintenance (the software doesn’t wear out, and changes functionality after repair; upgrading the software for 1000 cyber-physical platforms requires 1 action plus e-transmission vs. 1000 actions for hardware; but the software is
general much harder to understand and diagnose). For organizations continuing to maintain the software they developed, the performers often have a choice to continue with the software’s maintenance or to join a new development project. Some will identify with the current project’s objectives and wish to continue, but many will prefer to go to a new project to keep up with new technologies or to avoid the extensive technical debt remaining to be paid, leading to a loss of maintainer familiarity with the software. For enterprises with separate organizations and budgets for software acquisition and maintenance, complications can arise in getting maintainers paid to participate in software definition and development activities, particularly when the conspiracy of optimism already leaves the project underbudgeted. Even when maintainer participation budgets are available, the Vicious Circle will find that the maintainers are too busy compensating for the maintainability shortfalls in their current systems or, almost as bad, the maintainers will send the people that they can most easily do without. Besides increasing the level of participation of maintainers in the development process, the next-highest leverage will come from increasing the competence levels of the maintainers. An excellent detailed roadmap for doing this is the CMU-SEI People Capability Maturity Model [6]. As with the CMMs for systems and software engineering, it has a 5-level progression including such practices as staffing, work environment, training and development, compensation, workforce planning, career development, empowered workgroups, continuous capability improvement, and continuous workforce innovation.

C. Maintenance Methods, Processes and Tools (MPT) Shortfalls

The usual development endgame of the conspiracy of optimism is that the limited development budgets can pay for completing the needed mission capabilities or preparing the maintenance MPTs, but not both. These MPTs include release management procedures, architecture definition, diagnostics, version control support, testing support, defect closure status reports, and other forms of operational and development documentation. Trying to re-create or proceed without these aids is a major source of high maintenance costs. Another main source of high maintenance costs is the cost of maintaining compatibility with stovepipe systems that the system under maintenance needs to interoperate with. Many of these are independently evolving, and have made infrastructure commitments and underlying assumptions incompatible with the infrastructure and underlying assumptions of the system being maintained. Many of these sources could have been identified and addressed via more up-front system engineering, but generally are not due to inadequate system engineering resources. Calibration of the COCOMO II cost model found that added cost of rework and technical debt of minimal vs. necessary systems engineering is an exponential function of software size; an added 14% of project cost for a 10 KSLOC (thousands of source lines of code); 38% for 100 KSLOC, 63% for 1000 KSLOC, and 92% for 10,000 KSLOC [7]. A good example was a major project with 5000 KSLOC and a conspiracy of optimism budget of $2 billion. The project thought that it was doing well to allocate 30% of the budget ($600 million) for systems engineering, but the project’s actual cost was $8 billion, meaning that the allocation to systems engineering had been only 7.5%.

IV. A PROPOSED SOFTWARE MAINTAINABILITY READINESS LEVELS (SMRL) FRAMEWORK

The concepts of Technology Readiness Levels (TRLs) [8], Manufacturing Readiness Levels (MRLs) [9], and System Readiness Levels (SRLs) [10], [11] have been highly useful in improving the readiness of systems to be fielded and operated. But except for one Systems Readiness Level table [11] indicating that for Operations and Support, State of the Art systems have high training cost and lack of support; State of the Practice systems have training and support readily available, and State of the Obsolescence systems have high cost of maintenance and increased training cost, the current SRL content does not address system maintainability readiness. Given the discussions above on the Software Process Foresight Shortfalls and the Major Shortfall Categories, it appears worthwhile to develop and use a similar Software Maintainability Readiness Level (SMRL) framework to improve future systems’ continuing operational readiness and Total Cost of Ownership (TCO). Most likely, its content would also help on hardware-intensive systems or cyber-physical-human systems.

Table 1 on the next page provides a proposed SMRL framework. Its columns are organized around the three major maintainability readiness shortfall categories of Life Cycle Management, Maintenance Personnel, and Maintenance MPTs. In general one would expect a major defense acquisition project to be at SMRL 4 at its Materiel Development Decision milestone; at SMRL 5 at its Milestone A; SMRL 6 at its Milestone B; and SMRL 7 at its completion of Operational Test and Evaluation. Smaller, less-critical systems would be expected to be at least at SMRL 3 at its Materiel Development Decision milestone and at SMRL 4 at its Milestone A. Note that the SMRL framework emphasizes outcome-based maintenance incentives such as with Performance-Based Logistics [5][4] or Vested Outsourcing [12] at SMRL 7, and maintainability data collection and analysis (DC&A) at SMRL 8. Some good examples of DC&A-based improvements in reducing technical debt and TCO via modularization around sources of change and identification of product line commonalities and variabilities are provided in references [13][14][15]. Some other good sources of insight in reducing software maintenance costs are [16][17]. We are also conducting empirical studies of software maintainability metrics via analyses of open-source software systems [18].

Over-extreme forms of agile development have had difficulties with scalability as in [3], with security-critical and safety-critical systems, and with bridging incompatible infrastructures in multi-institution medical and crisis management systems. Some organizations have had significant successes in developing and evolving complex systems with DevOps.
and Continuous Delivery approaches, but generally with very highly skilled teams and enterprise-controlled interfaces and infrastructure. Chen’s paper [19], “Continuous Delivery: Huge Benefits, but Challenges Too,” is a good summary of the benefits and challenges.

V. EARLY EVALUATION RESULTS

The SMRL framework has been presented at three industry-government workshops, with useful feedback and iteration of the framework; and a major Aerospace Corp. project is preparing to apply it. Other evaluation results have included the evaluation of projects having high technical debt in both development and maintenance, and development of parametric models that relate the sources of technical debt to their ultimate magnitude. These include calibration of a model [20] to evaluate the return on investments in maintainability based on data from two TRW projects that did not make the investments and one that did: CCPDS-R, described in [21]. Another corroborative result is the analysis of exponential growth of technical debt due to systems engineering underinvestment experienced across the 161 projects involved in the calibration of the COCOMO II model’s architecture and Risk Resolution parameter [7][2]. The Vicious Circle phenomenon was exhibited in major architecture reviews of two large government projects. One project fortunately had two people with maintenance experience on the review team, who were able to provide maintainability recommendations that helped the project avoid significant maintenance costs. The other maintenance project did not have such people, and its maintenance organization experienced excessive workload growth and an inability to quickly and cost-effectively respond to needed changes.

The Conspiracy of Optimism phenomenon has been shown on several projects, in which the systems engineering budget was on the average 40% lower than the systems engineering cost estimated by the Constructive Systems Engineering Cost Model COSYSMO [22]. Combined with Hasty Contracting, two Conspiracy of Optimism systems of systems projects had poor performance with respect to rapid responsiveness to change. Their average time in workdays was 27 workdays to close a change request within an individual platform or capability group; 48 workdays if the change required coordination across multiple platform or capability groups; and 141 workdays if the change involved a change in performers contracts [20].

VI. CONCLUSIONS

Major and growing sources of savings in a system’s Total Cost of Ownership can be achieved via improvements in software maintainability. Some savings can be achieved by improving software maintenance via better Methods, Processes, and Tools (MPTs), but much greater savings can be achieved by addressing the root causes of increased maintenance costs due to overemphasis on initial acquisition cost-effective ness. These include shortfalls in Life Cycle Management aspects; Personnel capabilities and participation aspects, and MPT aspects. Drawing on the successful use of the Technology Readiness Level, Manufacturing Readiness Level, and System Readiness Level frameworks, this paper proposes a similar Software Maintenance Readiness Level framework, based primarily on cumulative improvement of the three acquisition shortfalls that result in increased maintenance costs. Several organizations are interested in experimentally applying it; one has an experimental application being prepared.

VII. ACKNOWLEDGEMENTS

This material is based upon work supported in part by the U.S. Department of Defense through the Systems Engineering Research Center (SERC) under Contract H98230-08-D-0171. SERC is a federally funded University Affiliated Research Center managed by Stevens Institute of Technology. It is also supported by the National Science Foundation grant CMMI-1408909, Developing a Constructive Logic-Based Theory of Value-Based Systems Engineering. We also acknowledge the support of National Natural Science Foundation of China No. 91318301, 61432001, 91218302.

REFERENCES

<table>
<thead>
<tr>
<th>SMR Level</th>
<th>OpCon, Contracting: Missions, Scenarios, Resources, Incentives</th>
<th>Personnel Capabilities and Participation</th>
<th>Enabling Methods, Processes, and Tools (MPTs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>5 years of successful maintenance operations, including outcome based incentives, adaptation to new technologies, missions, and stakeholders</td>
<td>In addition, creating incentives for continuing effective maintainability. Performance on long-duration projects</td>
<td>Evidence of improvements in innovative O&amp;M MPTs based on ongoing O&amp;M experience</td>
</tr>
<tr>
<td>8</td>
<td>One year of successful maintenance operations, including outcome based incentives, refinements of OpCon. Initial insights from maintenance data collection and analysis (DC&amp;A)</td>
<td>Stimulating and applying People CMM Level 5 maintainability practices in continuous improvement and innovation in, e.g., smart systems, use of multicore processors, and 3-D printing</td>
<td>Evidence of MPT improvements based on maintainance DC&amp;A based ongoing refinement, and extensions of ongoing evaluation, initial O&amp;M MPTs.</td>
</tr>
<tr>
<td>7</td>
<td>System passes Maintainability Readiness Review with evidence of viable OpCon, Contracting, Logistics, Resources, Incentives, personnel capabilities, enabling MPTs, outcome-based incentives</td>
<td>Achieving advanced People CMM Level 4 maintainability capabilities such as empowered work groups, mentoring, quantitative performance management and competency based assets</td>
<td>Advanced, integrated, tested, and exercised full-LC MBS&amp;SE MPTs and Maintainability other-SQ tradespace analysis</td>
</tr>
<tr>
<td>6</td>
<td>Mostly-elaborated maintainability OpCon, with roles, responsibilities, workflows, logistics management plans with budgets, schedules, resources, staffing, infrastructure and enabling MPT choices, V&amp;V and review procedures.</td>
<td>Achieving basic People CMM levels 2 and 3 maintainability practices such as maintainability work environment, competency and career development, and performance management especially in such key areas such as V&amp;V, identification &amp; reduction of technical debt.</td>
<td>Advanced, integrated, tested full-LC Model-Based Software &amp; Systems (MBS&amp;SE) MPTs and Maintainability-other-SQ tradespace analysis tools identified for use, and being individually used and integrated.</td>
</tr>
<tr>
<td>5</td>
<td>Convergence, involvement of main maintainability success-critical stakeholders. Some maintainability use cases defined. Rough maintainability OpCon, other SCSHs, staffing, resource estimates. Preparation for NDI and outsource selections.</td>
<td>Critical mass of maintainability SysEs with mission SysE capability, coverage of full M-SysE skills areas, representation of maintainability success-critical-stakeholder organizations.</td>
<td>Advanced full-lifecycle (full-LC) O&amp;M MPTs and SW/SE MPTs identified for use. Basic MPTs for tradespace analysis among maintainability &amp; other SQs, including TCO being used.</td>
</tr>
<tr>
<td>4</td>
<td>Artifacts focused on missions. Primary maintainability options determined. Early involvement of maintainability SCSHs in elaborating and evaluating maintainance options.</td>
<td>O&amp;M success-critical stakeholders provide critical mass of maintainability-capable SysEs Identification of additional M critical success-critical stakeholders.</td>
<td>Advanced O&amp;M MPT capabilities identified for use: Model-Based SW/SE, TCO analysis support. Basic O&amp;M MPT capabilities for modification, repair and V&amp;V: some initial use.</td>
</tr>
<tr>
<td>3</td>
<td>Elaboration of mission Operational Concept (OpCon), Architectural views, lifecycle cost estimation. Key mission, O&amp;M, success-critical stakeholders (SCSHs) identified, some maintainability options explored.</td>
<td>Highly maintainability-capable Systems Engineers (SysEs) included in Early SysE team.</td>
<td>Basic O&amp;M MPT capabilities identified for use, particularly for OpCon, Arch, and Total cost of ownership (TCO) analysis: some initial use.</td>
</tr>
<tr>
<td>2</td>
<td>Mission evolution directions and maintainability implications explored. Some mission use cases defined, some O&amp;M options explored.</td>
<td>O&amp;M integration, Life Cycle cost estimation</td>
<td>Initial exploration of O&amp;M MPT options</td>
</tr>
<tr>
<td>1</td>
<td>Focus on mission opportunities, needs. Maintainability not yet considered.</td>
<td>Evidence of needs for early expertise for maintainability, concurrent engineering, O&amp;M integration</td>
<td>Focus on O&amp;M MPT options considered</td>
</tr>
</tbody>
</table>