DOI:10.1145/2594413.2594422

How to cope with the growing demand for software solutions at no extra cost.

BY SHIMEON PASS AND BOAZ RONEN

Reducing the Software Value Gap

THE SOFTWARE VALUE gap is the unexploited potential of an IT division to increase the value of the overall organization. In today's dynamic business environment, most companies depend on value creation from software solutions delivered by IT.[a] These solutions are crucial for day-to-day running, controlling, and growing the business, as well as complying with regulatory requirements. In many cases, the availability of suitable software solutions is a prerequisite for launching new business initiatives and innovation.

In most companies demand for software solutions and functionality exceeds the IT budget (or capacity of the related human resources) for development and maintenance by up to 500%, especially when accounting for the "hidden queue" of software solutions.[9] This occurs since even the most prosperous companies cannot afford to allocate unlimited resources to IT, as it would adversely affect the value of the overall organization.[b]

a Value creation through a software solution is the marginal discounted cash flow originating from use of the solution.

b A company's value is its discounted cash flow.

Corporate boards and top executives decide periodically on an affordable IT budget, meaning business needs are only partially met. Moreover, the eventual software delivery flow is too small, late, and expensive. The software value gap is particularly disturbing in major industries (such as financial services, telecommunications, insurance, airlines, health care, and Internet retail), as well as in governmental agencies.

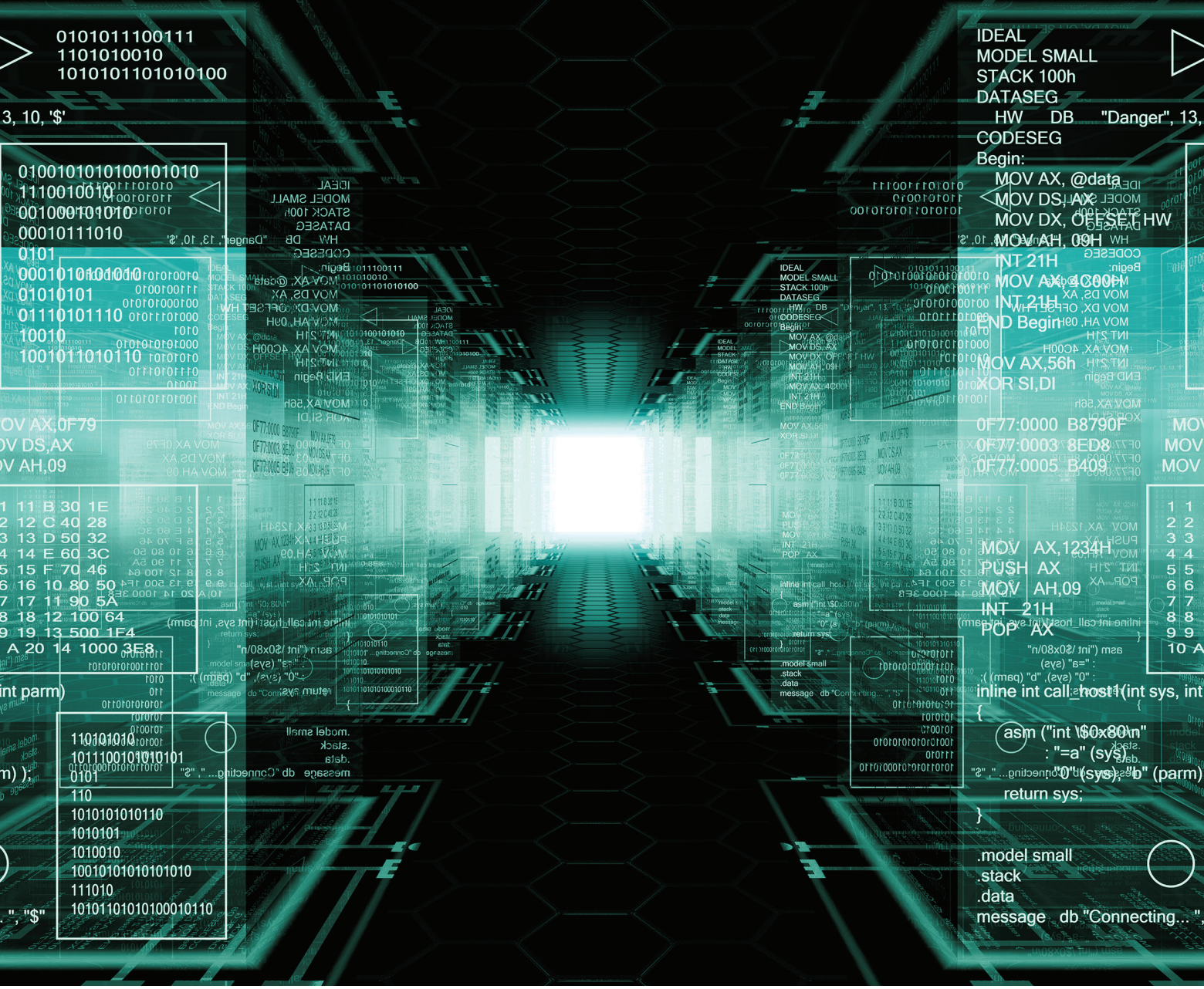
Conventional Approaches

One approach to reducing the software value gap is to invest in IT resources by hiring more IT employees and subcontractor employees, outsourcing projects to subcontracting companies (onshore and/or offshore), or purchasing off-the-shelf software packages. However, the extent to which companies are able to add more employees or budget is limited. Moreover, mere investment of additional money in IT is no solution. Outsourcing software development and purchasing software packages both involve allocating significant internal IT resources for requirements definition, systems analysis, integration with other software solutions, data migration, databases, data warehouses, implementation, and maintenance. In many cases, the extent of software-solution deployment is limited by the ability of the organization to define and agree on its needs, and later to implement, assimilate, and adopt new systems.

Another approach is to increase software development productivity by implementing one or more of the following methods:

» key insights

- **Software solutions delivered by IT divisions usually do not achieve full value creation potential.**
- **A comprehensive approach is needed to boost productivity and ensure IT value creation.**
- **Throughput enhancement and lead-time reduction can be achieved with simple managerial tools.**



- Agile, Scrum, and Extreme Programming development methodologies;[12]
- Critical Chain methodology and tools for reducing lead times of software development projects and improving due-date reliability;[1,7]
- Lean techniques to create a frugal IT organization;[2]
- DevOps techniques;[8] and
- Requirements management and software reuse.[3]

Yet another approach is to prioritize software development requests through some prioritization criterion. The extent these conventional approaches begin to close the software value gap is variable and usually not too great. Hence, they leave room for further improvement.

Scope of the Software Value Gap

To understand why software development and maintenance do not create enough value for the organization we list and analyze the generic problems associated with the typical IT development and maintenance environment. Analyzing them through the focused Current Reality Tree, or fCRT, points to the root causes of the software value gap.[11] We start by listing the generic undesirable effects, or UDEs, associated with software solutions development and maintenance,[6] in uncompromising language:

- Software-solutions development does not create enough value for the organization (the leading UDE);
- Inadequate software solutions productivity;
- Cost of the IT division too high;
- Delivered software solutions often abandoned;
- Software solutions in the "realization portfolio"[c] not maximizing value;
- No effective selection mechanism for software portfolio selection;
- Excessive ineffective time of IT developers;
- Requirements not properly defined;
- Software solutions that meet needs only partially;
- Lead times too long;
- Performance measurement lacking or misleading;
- No effective focus on value;
- Internal customers' involvement insufficient; and
- Lack of an efficient, effective methodology for managing IT resources.

Here, we arrange the UDEs in a fCRT (see Figure 1) where arrows indicate the causality relationship be-

c The realization portfolio consists of software solutions selected for development.

ILLUSTRATION BY MICHELANGELUS

Figure 1. fCRT of the software value gap.

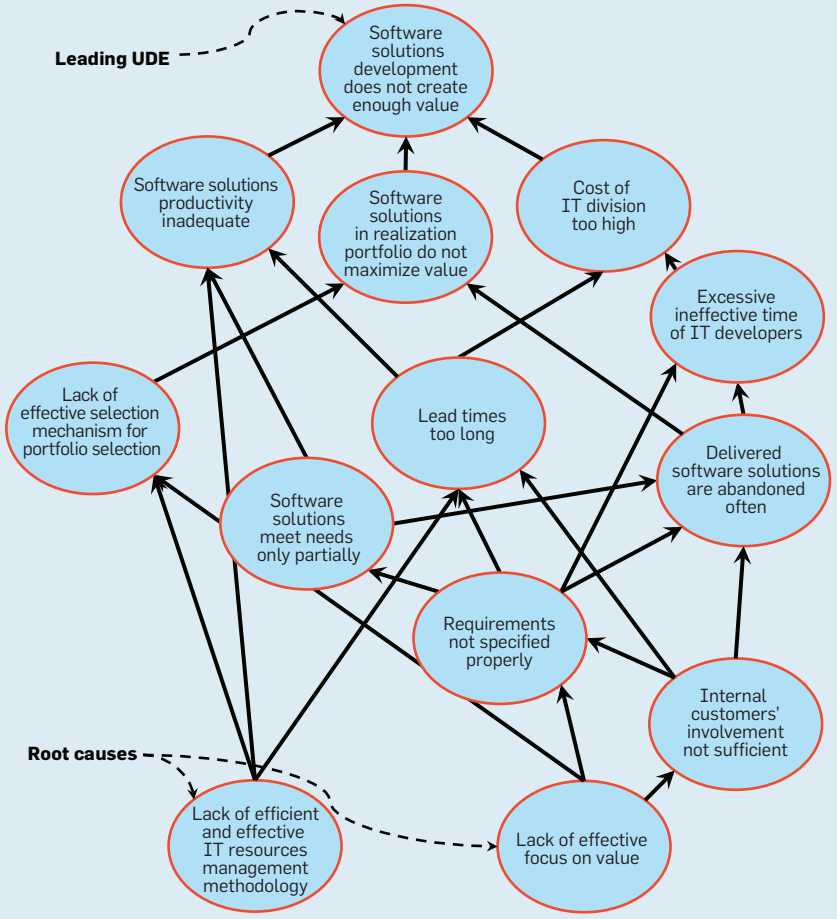
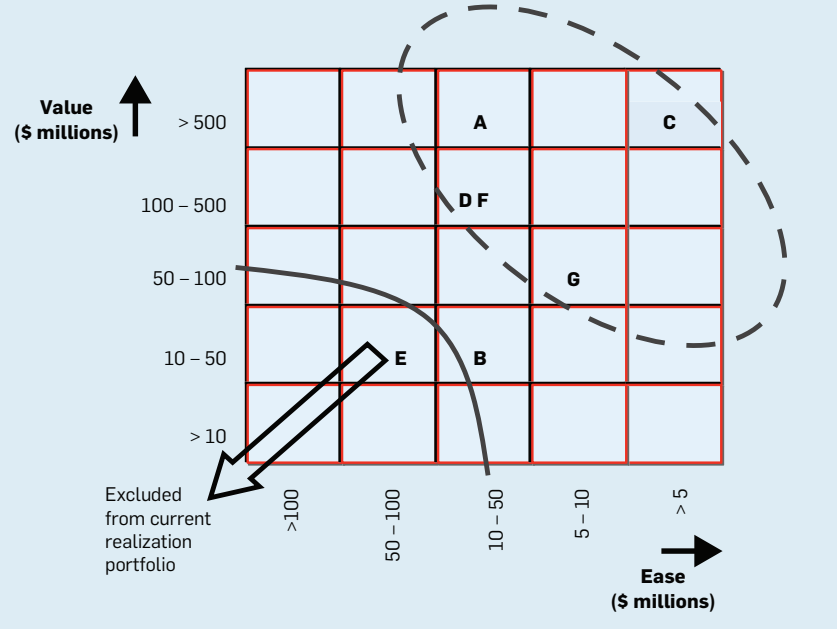


Figure 2. Focusing matrix for selecting software solutions (not to scale).



tween UDEs, pointing from the causing UDE to the resulting UDE. The process of fCRT building starts with putting the leading UDE on top. UDEs that are the causes of the leading UDE are then picked from the list and put below it and connected through the causality arrows. The process continues by adding more UDEs below the UDEs already in the tree that are causing them and subsequently adding the required causality arrows. The bottom UDEs of the fCRT are not associated with other UDEs causing them; they are thus considered the root causes for the software value gap.

This analysis identifies the two main root causes: lack of effective focus on value and lack of an effective IT resources management methodology.

Root causes of the software value gap are not just the result of IT division management practices but the behavior and norms within the organization as a whole. Reducing the gap means creating more value to companies and other organizations through better response to their business needs. This greater value can be realized by addressing two options: select the most valuable IT systems for development and maintenance (effectiveness) and enhance productivity in the IT division (efficiency). Here, we outline a comprehensive approach that enables a breakthrough toward reducing the software value gap through existing resources. Moreover, it synergistically complements the conventional approaches outlined earlier.

Focus on Value

Since demand for software solutions significantly exceeds supply, IT divisions are, by definition, permanent bottlenecks in their organizations and hence unable to satisfy all specified corporate IT needs.[d] Unfortunately, selection of projects for inclusion in the realization portfolio typically reflects the organizational power of the requesting division or the length of stay of the request in the to-do queue rather than real economic value for the organization.

d Permanent bottlenecks are resources that will remain bottlenecks, since demand is extremely large.

One way to change this counterproductive reality is to apply the strategic-gating process that results in a portfolio of software-development projects that maximizes value for the organization.[5] Counterintuitively, the portfolio leading to maximum value in many cases consists of only a small number of projects. The strategic-gating process consists of evaluation and selection that prioritizes IT requests according to their expected contribution to value creation relative to their IT resources requirement. It also ensures requests are well in line with the company's strategy and plans for growth.

The strategic-gating process is done annually or (preferably) quarterly. The requesting division provides for each software-solution request a value-creation estimate backed by a business plan, while IT experts provide a rough-cut estimate of their total cost of ownership, or TCO. Software solutions required by regulations are given an infinite value since they are mandatory.

It is advisable to exclude very small software solutions and change requests (CRs) from the strategic-gating process and put aside 15%–25% of the overall IT budget, allocating it among the various divisions. A high proportion of effort/budget dedicated to CRs usually increases internal users' satisfaction, while a large proportion of resources devoted to large- and mid-size software solutions contributes more to the organization's overall value. Determining the desired balance is a strategic decision.

Selecting software solutions can be done through the following procedure:

- Software solutions are listed and tagged with value-creation and TCO estimates (see Table 1);[11]
- Software solutions are mapped in a focusing matrix according to their value creation and TCO (see Figure 2);[11]
- Candidates for incorporation in the realization portfolio are selected mainly from the upper-right-hand quadrant of the matrix (highest value, lowest cost); and
- Selection continues until the available TCO budget for the relevant period is exhausted.

Value creation is also the appropriate criterion for approval of scope changes during delivery of software solutions. There is a spectrum of potential scope changes, from small modifications of features to substantial scope change or even a project swap. A scope change is justifiable if the result creates positive net value—the estimated increase in value creation of the software solution due to scope-change minus the cost of scope-change realization minus the cost of project disruption minus the cost of possible delay in delivery.

Strategic-Gating Process

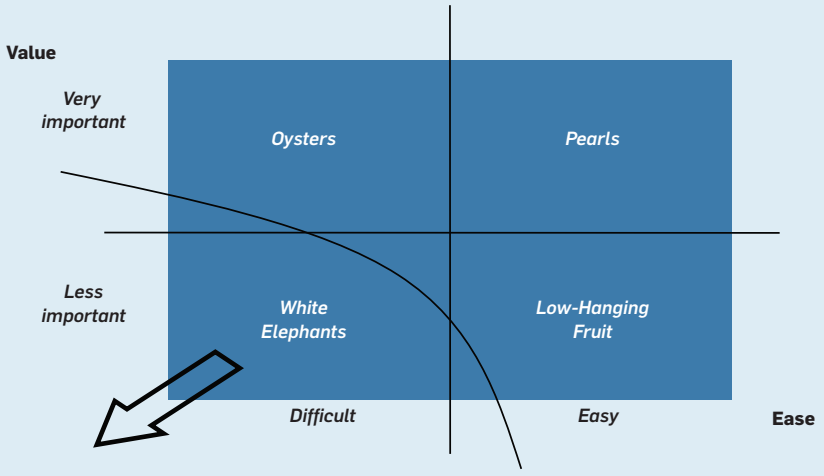
To demonstrate the numerical and graphical strategic-gating mechanism, consider an example in which seven software solutions are proposed by various corporate divisions for the strategic-gating process. The solutions' total estimated TCO budget is $188 million. Selection is required since the approved TCO budget for the forthcoming year is limited to $110 million; Table 1 lists the software solutions that are then mapped in a focusing-matrix according to their value creation and ease of realization, as in Figure 2. The most valuable software solutions reside mainly in the top-right-hand quadrant of the focusing matrix. Software solutions are picked by top corporate management with the aid of the matrix up to the total TCO limit of $110 million. In this strategic-gating process, corporate management picked software solutions A, C, D, F, and G as the realization portfolio.

Selecting software solutions for inclusion in the realization portfolio can also be achieved by ranking the software solutions according to their specific contribution, or ratio between the value of the software solutions and their TCO.[11] However, the focusing matrix allows management to adjust the portfolio according to strategic considerations and account for intangible aspects of the business and the market.

Table 1. Software solutions selection through the focusing table.

#	Software Solution	Value ($ millions)	Ease = TCO ($ millions)
A	Customer relationship management	550	45
B	Asset management	45	29
C	Bills transparency	Regulation	10
D	Call-centers knowledgebase	170	15
E	Human resources	25	55
F	Business intelligence upgrade	145	25
G	Campaign management upgrade	55	9
	Total TCO		188

Figure 3. Conceptual focusing matrix.



In the conceptual focusing matrix (see Figure 3), the top-right-hand quadrant is designated "pearls" since it contains the most valuable candidates for selection. The three other quadrants are designated "oysters" (valuable yet hard to crack), "low hanging fruit" (easily accomplished but less valuable), and "white elephant" (to be avoided). This terminology facilitates communication among managers.

Top management takes the final strategic-gating decision, deciding to include in the realization portfolio oysters if it views them as having strategic importance beyond the value-creation estimate. Likewise, top management could consider inclusion of low-hanging fruit if it considers it appropriate for achieving quick successes.

An outside observer might doubt whether division heads, being eager to promote their software-solutions requests, would deliberately exaggerate estimates of value creation. To obviate this risk, every request for a software solution must be backed by a business plan (or at least a mini-business plan). In addition, the division heads must commit to the anticipated value creation and consequently include the derived cash-flow estimates in their revenue targets. A follow-up mechanism concerning how to achieve these value-creation targets should be linked to the organization's key performance indicators and incentive systems.

To be effective, performance measures must be properly linked to the goal of value creation.

Waste Can Be Avoided

Research shows at least 50% of IT developers' time is wasted[11] due to the following reasons:

- Rework due to incomplete or poorly defined needs and requirements ("incomplete kit");
- Rework due to frequent changes in requirements and scope up to the final delivery stages; most such changes are not "must have" but only "nice to have";
- Software solutions developed and delivered but eventually not used (happens all too often);
- Over-specification of requirements to include functionality and features seldom or never applied; and
- Having too many activities assigned to individual developers, leading to wasteful context switching among activities (bad multitasking).

Waste cannot be totally eliminated but can be reduced significantly through seven simple managerial practices, or remedies:

The complete-kit concept.[e] To avoid waste of IT resources due to rework, the organization, as a whole, should adopt and implement the complete-kit concept.[10,11] Projects should not be approved for development if their business rationale, requirements, business plan, or statement of work are only partial or vague. Likewise, tasks should not be assigned to individual developers if the requirements/specifications/design are incomplete or fuzzy. The content of the complete-kit definitions for the requirements and specifications is listed jointly by the requesting divisions and IT personnel.

Since requesting divisions must provide complete kits of requirements and specifications, they must understand their needs and the required software solutions. This would minimize the extent of requirements changes with resulting rework and the number of projects developed and delivered but eventually abandoned. Moreover, implementation of the complete-kit concept usually leads to improved communication and collaboration between business divisions and the IT division.

Also needed are definitions of complete kits for tasks and activities performed along a project's life cycle; for instance, the system-analysis files given to programmers by their managers must contain a complete kit of information to enable them to do their jobs properly. Likewise, files given to testers must contain the complete kit of information to enable testers to complete their jobs properly.

Eliminating over-requirements, over-specifications, overdesign. Scrutiny of software-solution requirements usually reveals a high degree of over-requirements. Significant portions of functionality and features required for a software solution are nice to have rather than must have. In many cases IT professionals tend to introduce over-specification and overdesign to be on the safe side in

e The complete-kit for a task is the list of all items required to complete the task without interruption.

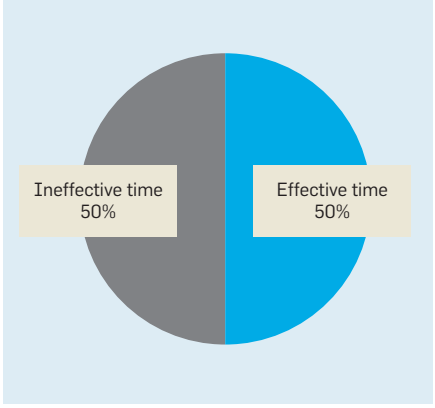
their view or to make the project more challenging. Coman and Ronen[4] estimated the amount of work-force time wasted over these phenomena exceeds 30%. Needed is a change in attitude. Managers and programmers alike, in and out of IT, are responsible for challenging functionality and features they view as over-required, over-specified, or overdesigned. These issues should be raised multiple times during the delivery cycle. For example, during the kick-off meeting, over-requirements should be identified and eliminated from the scope of the system to be developed. In team meetings concerning risk management, control gates, and design reviews, over-requirement, over-specification, and overdesign should likewise be identified and eliminated.

25/25 practice. The 25/25 rule says management should attempt to discontinue and stop work on approximately 25% of the projects in the pipeline. In the remaining projects, unneeded or over-required features (approximately 25%) should be removed. All software solutions in the pipeline should be examined on a quarterly basis by top management through the focusing matrix. Business situations might have changed, and value creation might be lowered significantly. Likewise, for some software solutions the remaining delivery costs end up being much higher than expected. In such cases, where projects lose their value-creation potential, top management must stop project delivery, disregarding the "sunk costs" already invested in them. In some organizations, many projects can be eliminated this way, freeing up to 25% of the IT division's budget or capacity.

Similarly, the team must scrutinize most complicated and costly software solutions remaining in the pipeline to detect over-requirements, over-specifications, and overdesigns and eliminate them. This practice removes unnecessary functionality and features, reducing the cost of delivering these solutions by up to 25%.

Split large and risky solutions among releases. In organizations where software is launched in periodical (such as quarterly) releases, we recommend refraining from developing and implementing large software solutions in a single release, especially when delivery involves significant business and technical uncertainty. Splitting the software solution, if possible, among two or three consecutive releases ensures the requesting division gains a better understanding of its needs and solution requirements; technical risk is also reduced. This practice yields a better fit with business needs while reducing the hazard of rework due to changes in requirements and the probability of eventual neglect of the software solution.

Figure 4. Effective vs. ineffective time (before improvement).



Performance measurement. Systematic measurement of performance is a powerful, proven means to enhance performance, including avoidance of waste. However, to be effective, performance measures must be properly linked to the goal of value creation.

Performance measures are not supposed to be "perfect" or "scientific" or cover all extreme cases. Defining perfect measures is generally difficult or impossible. The main purpose of performance measures is to enable improvement over time. Measures that are not perfect yet make sense and are measured in a consistent manner over time are good enough. Though other performance measures can be added as long as they are in line with the organization's overall business goal, we suggest a basic set of seven periodical performance measures covering most operational aspects of IT:

Throughput of IT division. T = total estimated value creation of software solutions delivered during the measurement period;

Productivity of IT division. Prod = the amount of CR-equivalent units developed during the measurement period; a possible definition is the total of {number of large software solutions multiplied by 9 + number of medium-size software solutions multiplied by 3 + number of change requests} delivered during the measurement period;

Operating expenses. OE = TCO expenses for IT during the measurement period;

Work in process. WIP_1 = number of software solutions open in development at the measurement instance; WIP_2 = average number of released activities per developer in the IT division at the measurement instance;

Lead time. LT = average time span from requirements introduction until delivery for all large software solutions during the measurement period;

Quality. Q_1 = number of critical defects detected during the first six months following delivery for all software solutions; Q_2 = average scope stability[f] of all software solutions delivered during the measurement period; and

Due-date performance. DDP_1 = percentage of software solutions delivered on time during the measurement period; DDP_2 = percentage of development activities delivered on time during the measurement period.

These measures are solid performance indicators and useful in creating an effective incentives system for the head and managers of the IT division.

Short lead times. Long lead times calling for over-requirements and unnecessary changes in scope can be prevented by substantially shortening project lead times through practices we discuss later.

Net value creation for scope-change requests. To prevent introduction of unnecessary requests for scope change, the approval criterion is the existence of positive net value creation discussed earlier.

These seven remedies help avoid waste within an IT division. To understand their potential for productivity enhancement consider the following example IT division whose professionals experience approximately 50%

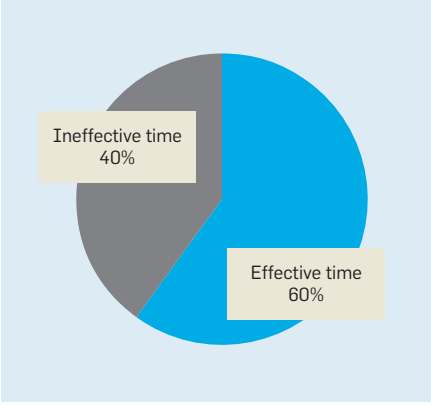
f Scope stability reflects the extent of changes introduced into the scope definition of the software solution.

waste of their work-time capacity due to several sources (see Figure 4). Suppose by applying one or more of the remedies to major waste sources, waste is not totally eliminated but rather conservatively cut to 40%. This means productive time actually grows from 50% to 60% with the same resources (see Figure 5). Increasing effective time from 50% to 60% is like adding 20% trained and experienced resources at no extra cost in terms of, say, wages, recruitment, training, mentoring, working space, workstations, or software licenses.

Controlled Release

A proven practice for improving IT efficiency is to control the release of projects into the system, as well as the release of tasks to individual developers, enabling fast flow of work through the system, shorter lead times, and increased productivity. At the same time, it allows better utilization of the bottleneck resources of the IT division.

Figure 5. Effective vs. ineffective time (following waste reduction).



High-level tactical gating controls the release of projects onto the development floor according to a predefined prioritization mechanism; starting too many projects at once could bring chaos, so staggering projects is preferred. Low-level tactical gating is a mechanism for controlling the release of tasks to developers according to a predefined prioritization mechanism. Managers and team leaders release new tasks to their subordinates only if they have fewer than three or four released tasks, or two to four weeks of work. The requirements/specifications/design of these tasks must also comply with the complete-kit policy. Instead of assigning large tasks to a developer, we recommend dividing large tasks into smaller activities taking five to 10 days each. In addition, according to low-level tactical gating, IT managers should spare bottleneck developers as much as possible day-to-day interruptions like customer support and unnecessary meetings. Synchronized implementation of high- and low-level tactical-gating mechanisms results in shorter lead times and faster flow of projects, as well as improved due-date performance and software quality.

Telecom Case Study

B is a multibillion-dollar telecommunication company whose business depends on its IT division in all activities, including sales, marketing, operations, engineering, customer service, billing, and finance. B's top management has decided to implement the value-creation methodology and tools throughout the organization. It also participated in a six-day seminar covering the value-creation managerial philosophy, methodology, and tools, including presentations, discussions, and hands-on assignments. At the end of the seminar, top management approved an implementation plan for the following six major value drivers, as well as for the IT division itself:

- Identifying and managing IT division bottlenecks;
- Implementing strategic gating and 25/25 mechanisms;
- Implementing the complete-kit concept in the main work processes of the division and its subcontractors;
- Implementing high-level and low-level tactical gating while eliminating over-requirements, large activities, and bad multitasking;
- Avoiding ineffective times; and
- Defining performance measurements for the IT division.

All managers in the IT division, up to team leaders, participated in follow-on three-day seminars covering value-creation enhancement, forming several task force teams to address value drivers.

Top management reported three main results (within three years):

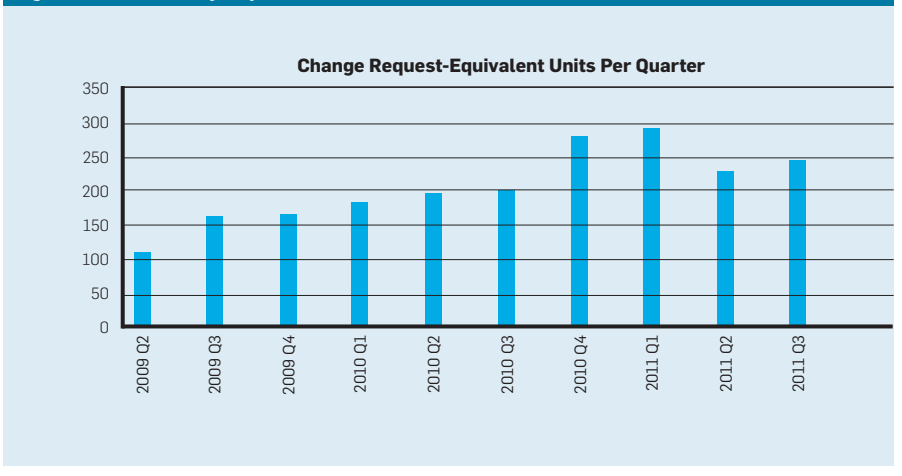
Productivity. Increased 120% from 109 CR-equivalent units to 241 CR-equivalent units per quarter (see Figure 6);

Operating expenses. Annual TCO budget practically constant; and

Due-date performance (DDP_1). Increased from 69% on-time delivery to 76% on-time delivery.

We found similar results in seven other companies we studied.

Figure 6. Productivity improvement over time.



Value Leverage

Predicting the effect of the strategic-gating mechanism is difficult, as it is situation-dependent, reflecting the actual value and ease of all potential software solutions proposed for a specific situation and the realization portfolio that would have been selected by the IT division in the absence of a strategic-gating mechanism being in place. However, the effect of implementing IT productivity improvement on the value of the company can be clearly shown.

Value-leverage example. Consider a company in which approximately 30% of revenue depends on or originates

from new software solutions. The previous year's revenue totaled $1.29 billion. Its real variable costs (RVCs) were $210 million (16.3% of revenue). The resulting throughput (all revenues minus RVC) was $1.08 billion. The company's fixed costs totaled $910 million, hence its earnings before interest, tax, depreciation, and amortization, or EBITDA, the previous year was $170 million (see Table 2).

Now suppose the company can consistently achieve annual revenues of $1.29 billion. If we use a conservative EBITDA multiplier of 10, the company's market value is $1.70 billion.

Suppose the IT division succeeds in implementing the kind of improvement practices described here, thereby increasing the amount of software solutions delivery by 20%. Suppose, too, that the value-creation potential of these additional 20% software solutions is on average only 25% compared to average projects in the current realization portfolio. Since only 30% of the revenue originates from the introduction of new software solutions, the additional revenue for the following year would be $1.29 billion multiplied by 30% multiplied by 20% multiplied by 25% = $19 million. RVC will probably increase proportionally to $213 million, or still approximately 16% of revenues.

If the IT division achieves this increase in productivity of software solutions with the same resources, and does it without adding human resources or assets to the company, then fixed costs are unchanged. Our own calculation shows the corporate EBITDA would grow to $186 million, a 9.4% increase over the previous year (see Table 3). Using the same EBITDA multiplier of 10, the market value of the company reaches $1.86 billion without adding significant resources or other investment. Our experience shows that implementing the strategic-gating mechanism adds even more value to the company overall.

These improvement steps require a change in an organization's overall culture, thus the leadership of the CEO, CIO, and top management team. The strategic-gating mechanism and IT division's internal improvement activities are synergic and must be implemented concurrently. If the business divisions have doubts regarding the IT division's commitment to introduce required improvements, they would be reluctant to participate in the strategic-gating selection mechanism or submit project requests in the form of complete kits. Likewise, if IT management does not define requests according to the complete-kit concept free of over-requirements, it will have little motivation to improve its own processes. The culture of splitting large software solutions into "stages" or "releases" can be introduced at a second stage, once the other improvement steps are in place.

Table 2. Profit-and-loss summary.

	Last year ($ millions)	
Revenue	1290	
Real variable costs	210	[16.3%]
Throughput	1080	
Fixed costs	910	[70.5%]
Earnings before interest, tax, depreciation, and amortization	170	[13.2%]

Table 3. Value creation, as seen in the profit-and-loss summary.

	Last year (million $)		Next year (million $)	
Revenues	1290		1309	
Real Variable Costs	210	[16.3%]	213	[16.3%]
Throughput	1080		1096	
Fixed costs	910	[70.5%]	910	[69.5%]
EBITDA	170	[13.2%]	186	[14.2%]
Δ(EBITDA) [~ΔValue]			+9.4%	

Conclusion

This approach to reducing the software value gap complements conventional approaches outlined here through a high degree of synergy. Its ability to add value has proved to be achievable without further investment. Improvement is possible in several months. Success along these lines is easier to accomplish when the initiative for the improvement project comes from the CEO or the board of directors and the CEO becomes the "owner" of the project. Reducing the software value gap for enhanced value creation complements other value-creation activities when other sectors of the organization (such as marketing, sales, R&D, engineering, project management, and operations) pursue them.

References

1. Anderson, D.J. *Agile Management for Software Engineering.* Prentice-Hall, Upper Saddle River, NJ, 2003.
2. Bell, S.C. and Orzen, M.A. *Lean IT.* Productivity Press, New York, 2011.
3. Boehm, B.W. *Software Engineering Economics.* Prentice-Hall, Upper Saddle River, NJ, 1981.
4. Coman, A. and Ronen, B. Icarus' predicament: Managing the pathologies of overspecification and overdesign. *International Journal of Project Management 28*, 3 (Apr. 2010), 237–244.
5. Cox, J.F. III, Boyd L.H., Sullivan T.T., Reid R.A., and Cartier, B. *The TOCICO Dictionary, Second Edition.* McGraw-Hill, Inc., New York, 2012.
6. Goldratt, E.M. *It's Not Luck.* North River Press, Croton-on-Hudson, NY, 1994.
7. Goldratt, E.M. *Critical Chain.* North River Press, Croton-on-Hudson, NY, 1997.
8. Loukides, M. *What Is DevOps?* O'Reilly Media, Inc., Sebastopol, CA, 2012.
9. Martin, J. *Applications Development Without Programmers.* Prentice-Hall, Englewood Cliffs, NJ, 1982.
10. Ronen, B. The complete kit concept. *International Journal of Production Research 30*, 10 (1992), 2457–2466.
11. Ronen, B. and Pass, S. *Focused Operations Management.* John Wiley & Sons, Inc., Hoboken, NJ, 2008.
12. Schwaber, K. *Agile Project Management with Scrum.* Microsoft Press, Redmond, WA, 2004.

Shimeon Pass (shimeon@passmgmt.com) is a senior consultant and implementation facilitator at Focused Management Ltd., Tel Aviv, Israel.

Boaz Ronen (boazr@post.tau.ac.il) is the Professor Simon I. Domberger Chair for Innovative Value Creation and a professor of technology management and value creation in Recanati Business School, Tel Aviv University, Israel, and a senior consultant at Focused Management Ltd., Tel Aviv, Israel.

Copyright held by Author/Owner(s). Publication rights licensed to ACM. $15.00

Copyright of Communications of the ACM is the property of Association for Computing Machinery and its content may not be copied or emailed to multiple sites or posted to a listserv without the copyright holder's express written permission. However, users may print, download, or email articles for individual use.