Information Systems Security, 16:80–89, 2007

ISSN: 1065-898X print/1934-869X online
DOI: 10.1080/10658980601144899

Taylor & Francis
Taylor & Francis Group

Effective Change Management: Ensuring Alignment of IT and Business Functions

William A. Yarberry, Jr.
President, ICCM Consulting,
Houston, Texas, USA

INTRODUCTION

Change management is a core information technology general control required to support the business functions of any enterprise. While change control is conceptually simple, the mechanics of implementation and monitoring require attention to detail as well as support from IT, users, and business unit management.

At its most basic level, change management is a control system that ensures programs, systems, and infrastructure modifications are authorized, tested, documented, and monitored. One layer below these simple objectives is a plethora of details that challenge even the most procedurally oriented IT organization. As a first step towards good practices, the enterprise needs to set up objectives at the policy level. Examples include:

- Application and infrastructure changes are properly approved, both for work initiation and later migration to production.
  Proposed changes are prioritized based on business needs.
- Changes are auditable and can be traced "up and down" the process. For example, a variation in the size of an executable module can be traced backwards to documentation authorizing a change; conversely, an authorization for a specific change can be linked to detailed code modifications.
- Failed changes can be rolled back.
- Changes to application code and configurations are tested and approved prior to implementation in production.
- Users participate in application related testing of changes.
- Segregation of duties is maintained. Developers do not promote code into production, and "move specialists" do not have access to source code/libraries.
- Procedures exist to ensure urgent/emergency changes are implemented in a controlled and auditable manner.
- Changes are "sized" so that the level of testing and review is appropriate, given the financial/operational impact of the change.
- The process for requesting a change is standardized and subject to known procedures. For example, direct requests for changes from users to developers via phone calls are not acceptable.

Address correspondence to
William Yarberry,
2903 Kings Forest Dr.,
Kingwood, TX 77339.
E-mail: byarberry@iccmconsulting.net

This article originally published as "Change Management" in EDPACS, 2005, 33(4):12–24.

The objectives listed above are examples that are common across most organizations. Individual firms may choose to link other requirements to the change management process, such as completion of required steps in a systems development life cycle (SDLC).

Implementation of effective change management is part of the worldwide movement towards improved IT governance and transparency. In the United States, for example, PCAOB auditing standard 2 specifically mentions program change management as a key control element.

THE CHANGE MANAGEMENT CYCLE

Figures 1A and 1B show the key elements of the change management cycle. The organization's SDLC is in the middle of the standard (non-emergency) process. Most of the work occurs within the SDLC box—design, development, testing, documenting, and obtaining approvals. Conceptually, change management serves as two bookends supporting a shelf of books representing development, testing, documentation, and approvals. By the time the SDLC process is complete, all the artifacts (documents, online documentation, etc.) should be in place. The sizing (risk assessment) process drives the artifacts required; for example, a low-risk change would rarely require an integration test, whereas a large ERP installation (sized at high risk) may require all the defined artifacts.

The individuals serving as change control managers need clear guidelines to answer questions such as:

- How is a change ranked (sized)? By number of staff hours required to complete the project? By intimate knowledge of the application and the

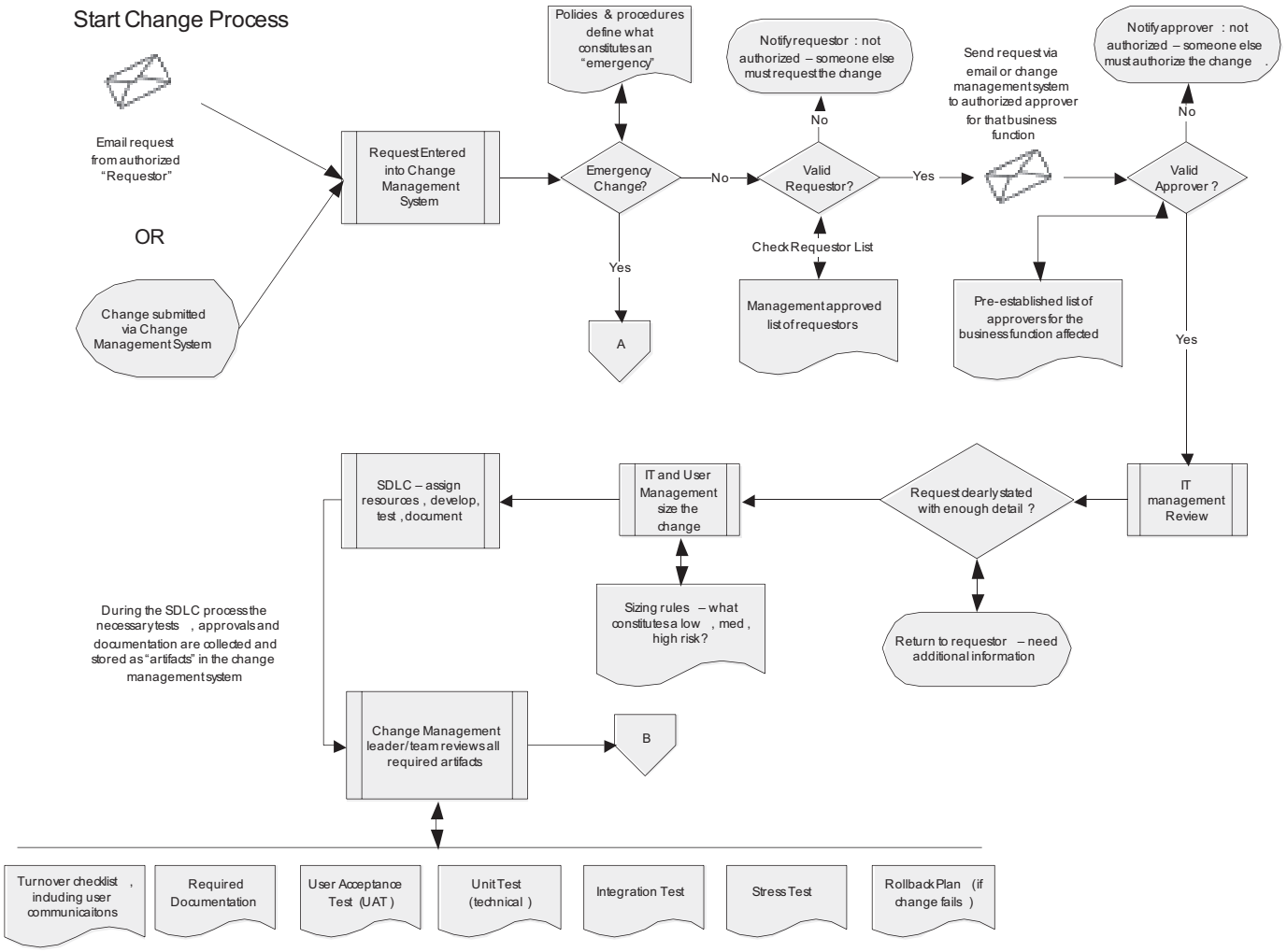

FIGURE 1A **Example Change Management Process**

likelihood that any given change will affect customer service? Each organization's rules for sizing will vary.

- Who can request a change? Ideally, there should be a separate requestor and approver for any change. Beyond that, the requestor should be someone knowledgeable about the application. Uninformed requests waste resources. The fact that an individual works in accounting does mean he or she is qualified to request, for example, specific changes in creditworthiness calculations.
- Who can approve a change, and who approves if that individual is not available? Typically a department head or leader would be listed as an approver, with a more senior person serving as an alternate. In no case should the requestor be the same as the approver. See the segregation of duties discussion later in this article.

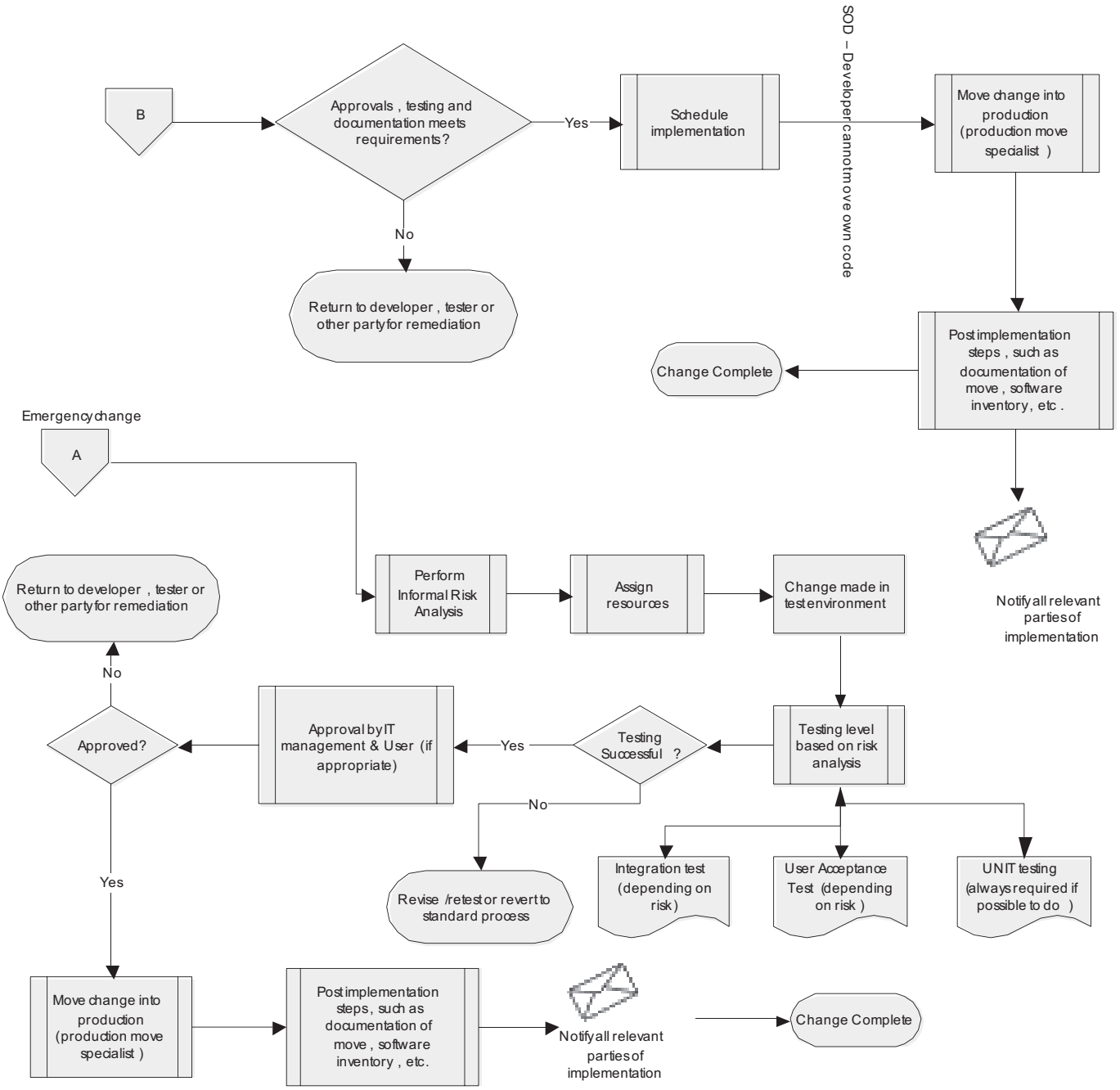

FIGURE 1B **Example Change Management Process (continued)**

- What is an emergency? Change management policies should clearly define the criteria for an emergency change. Since emergency changes receive less scrutiny and may occur at odd hours, the process to handle them should be clearly defined. Also, the term "emergency" is not related to "priority." Some organizations define emergency changes as those necessary to fix something that literally stops working or suddenly creates incorrect results (it worked yesterday but does not work today). Others may define them as a change required to satisfy an urgent, immediate, and unanticipated business need which, if not satisfied, will result in significant loss to the organization.
- What are the artifact requirements? The end of the change management cycle is analogous to a day in court. The "jury" is presented exhibits showing what happened during the systems development life cycle—was document #030.10, emergency backout plan, prepared in accordance with policy? Are results from the user acceptance testing attached to the change management ticket? The change control manager can only make these assessments if the requirements are clearly documented. For example, an integration test is typically not required for an enhancement to a sales report.

CHANGE CONTROL AS THE "ENFORCER"

Robert Burns' observation that the best laid plans often go astray applies equally well to systems development. There are many roadblocks to success: the schedule for each phase may become compressed as market forces escalate the priority or legislation driven deadlines get closer. Approvals, testing, or documentation may be omitted or reduced in scope; busy developers may regard some of the steps as bureaucratic impediments to productivity. One counter to these shortcomings is a robust change control system. It serves as the final checkpoint to validate compliance and to prevent promotion of changes that are unapproved, improperly tested, or inappropriately scheduled. A good change control system helps ensure that the sins of the developer will not see the light of day—bad code or configuration changes will be rejected.

TABLE 1 **Sizing Categories for Changes**

Size or Risk Ranking	Estimated Hours	Estimated Cost
Low risk/small project	<100	<$15,000
Medium risk/ medium project	100–1,000	$15,000–$150,000
High risk/large project	>1,000	>$150,000

Sizing and Risk Ranking

An effective change management system requires *just enough* approval, testing, documentation, and review. For example, a report heading change does not require the same level of review as a change in depreciation method. A sizing and risk ranking process helps IT respond to business needs quickly by minimizing the work required for small changes and strengthening requirements for high-risk changes. Table 1 shows a typical sizing/risk ranking scheme (it will vary considerably by organization).

The individual or committee performing the initial sizing may override these mechanical guidelines if, for example, a small change represents a significant risk to the organization. The purpose of the sizing is to assess the artifacts (tests, approvals, documentation, etc.) required for the change to be allowed to move into production.

Fortunately, systems changes follow Pareto's principle; 10-20 percent of the changes represent 80-90 percent of the risk. Since most of the changes are low risk, the need for speedy response to business requests can be met while the higher risk changes/projects receive appropriate scrutiny. To correctly size projects, the organization needs to do three things:

1. Size the change, using the expertise of the developer(s) *and* an IT manager(s). This roughly catalogs the proposed change in a "small" versus "medium/large" buckets.
2. Determine whether additional sizing effort is needed. For example, a seemingly innocuous change may affect downstream applications. For projects that may require considerable resources to complete, a better sizing allows a better return on investment calculation. Since it takes some resources just to complete the sizing, this step is necessary for complex changes.
3. Re-evaluate the sizing if the project is more complex than originally thought.

Generally, developers do not savor paperwork, even if the forms can be submitted electronically. Hence, there is a temptation to classify changes as smaller than they really are; a small sizing requires less review, fewer online forms, and a speedier delivery to the user. The person doing the work may contribute technical and application background information but should not be responsible for the final risk ranking.

ARTIFACTS

Before any change goes into production, the existence and quality of artifacts must be reviewed. Going beyond traditional systems documentation, artifacts detail the life history of any change or project. They answer questions such as "what is the evidence of testing, approval, financial analysis, operational review, and conformity to technical standards?" During the review of artifacts, a well-intentioned drive to perfection may arise. Impractically high standards weaken the change control system. During the implementation of a change management system, it is easy to list requirements and label them *all* as mandatory. To use an analogy, generals may easily move pins around on a map, but the troops on the ground must go through the formality of "making it happen." The artifacts required should be as minimal as possible, consistent with the organization's appetite for risk. More depth is certainly welcome, but if requirements are excessive, the system will break under the day-to-day pressures of production. *Reasonable* standards persist.

Following is an example list of artifacts. These will vary based on the organization's SDLC and special needs. Change management is not limited to control and risk issues. The CIO may have a specific initiative, such as marketing meetings with users, she wants to ensure take place. These initiatives take their place on the checklist, along with backout plans, as required artifacts.

- Scope statement
- Financial assessment
- Technical assessment
- Functional specifications
- Business requirements
- Project plan
- High level design
- Unit test plan
- Integration test
- Regression test
- Stress/capacity test
- User acceptance test
- Production turnover checklist

Testing

Few would argue the need for testing a change or new system. The challenge is to perform the right kinds of tests (unit, integration, and regression) so that reasonable efficiency is maintained without undue risk. The first step is to develop a matrix that relates the risk of the change to the level of testing. Table 2 presents an example risk based testing matrix.

TABLE 2 **Testing Requirements Based on Risk**

Change Type	Change Description	Unit	Integration	Regression	UAT*
Low risk	Affects only one system; easy to backout; minimal impact if it fails	Y	N	N	Y
Medium risk	Crosses multiple applications; could affect operations and financials	Y	Y	N	Y
Heavy risk	Crosses multiple applications; backout process extensive; could affect entire network; major financial/operational impact	Y	Y	Y	Y
Infrastructure only	Changes to hardware, middle ware, operating system and other IT elements not directly related to applications	Y (if possible)	Y (if applicable)	N	N

* User Acceptance Testing

Testing phases vary somewhat in scope from one organization to another, but generally the following are included:

- *Unit test:* Code within a program or module is tested with sample data and simple scenarios. For example, a firm may decide to change sales commissions from 10.5 percent of sales price to 11.0 percent. The developer runs the modified program in a test environment, compares results to hand calculated totals, and saves a screen print of the results as proof that he has unit tested the change.
- *Integration test*: A change is tested in both the application affected as well as downstream systems. For example, a change in a human resources system could result in an incorrectly formatted parameter file that feeds a data warehouse used by other applications. The integration test runs transactions through the application all the way to the last downstream application that would reasonably be affected. In most cases, only the immediately succeeding modules after the changed application need be included. Professional judgment must be used to determine whether the third or fourth applications in line should be tested.
- *Regression test:* A complex and typically time-consuming "test deck" that includes specific transactions considered key to the proper functioning of the software. For a full scope enterprise resource planning (ERP) system, a regression test might include more than 100 individual tests, such as "apply a credit to accounts receivable using a customer number known to be incorrect and then note whether it is rejected." Creating a thorough regression test a significant effort for both IT and user personnel. Of course, the payoff is significant. A portion of a sample regression test is shown in Table 3; large production systems will typically have many more steps. The next test, user acceptance, is the final checkpoint before production migration.
- *User acceptance test (UAT)*: Compiled solely by the business user, the UAT uses business terms to describe what any change should accomplish. This test focuses specifically on expected results particular to the change. To support Sarbanes-Oxley section 404 compliance requirements, test results should be maintained in the change management system. Screen prints, reconciliations, and other summary results are typically required; massive binders of paper showing line item testing are not normally required to show compliance.

"Hybrid" Elements

The classical focus of change management includes lines of code, objects, schemas, and other components of the organization's core applications. However, many large systems, such as ERPs (e.g., Oracle and SAP), can be modified significantly by setup changes that are solely under the control of the end user. For example, an appropriately authorized user in the finance or accounting department could set all new fixed assets to be setup with double declining balance as the depreciation method rather than straight line. Traditional change control processes would not include a review of these changes because they are not implemented by the IT group.

One point of view is that such changes are no different than any other major business decisions, many of which are made without consultation with the IT group. Clearly, controls within the business units should include segregation of duties and second-level review for any major accounting or policy change. However, a more realistic perspective recognizes that systems have grown increasingly complex and parameter/setup modifications can change processes in ways not contemplated by the change initiator. While users may have a thorough understanding of the system functionality for a particular part of the business, they may not have occasion to learn parts unrelated to their day-to-day duties.

In contrast, the IT group is charged with maintaining the entire system and so may be aware of the "downstream" effect of a major parameter change. For medium to major risk changes, a *dual* user-IT review is usually optimal.

Patches and Releases

The inner architecture of in-house written code is often, though not always, reasonably well known. Any changes to be introduced are likely understood in the context of the entire system. In-house changes could potentially be divided or rearranged

TABLE 3 **Partial Sample of Regression Test**

Section	Step	Prerequisite	Test Case Description	Module	Lead	Estimated Hours	Status
1. Define Items	1.1		Define new item category in the Inventory Category Set for use in the new item	Inventory	Jane D., John P.	0.20	Pass - 5/18
	1.3		Define a new item using template	Inventory	Jane D., John P.	0.10	Pass - 5/18
	1.4	1.3	Update master controlled attributes (i. e., COGS and Sales accounts)	Inventory	Jane D., John P.	0.10	Pass - 5/18
	1.5	1.3	Enable item in child organization and apply Org level template	Inventory	Jane D., John P.	0.15	Pass - 5/18
	1.6	1.3	Update item category set with new category	Inventory	Jane D., John P.	0.10	Pass - 5/18
	1.7	1.3	Update organization attributes (i.e., Planner and Lead-time)	Inventory	Mike T.	0.10	Pass - 5/18
	1.8	1.3	Enter item standard cost for Buy part in inventory organization (Dept 45)	Inventory	Charles G., John P.	0.22	Pass - 5/18
	1.9	1.3	Print Item Definition Detail Report	Inventory	Jane D., John P.	0.25	Pass - 5/18
	1.11	1.6	Create and verify Z43 Planned Safety Stock	Inventory	John D., Alfred A.	0.75	Passed
	1.12	1.6	Create and verify an Inventory Planned Safety Stock	Inventory	Shawn K., Po W.	0.25	Passed
	1.13	12.1	Cycle Days Supply update program	Inventory	Alfonzo W.	0.50	Passed on retest
	1.14	12.1	Cycle Post Processing Lead-time update program	Inventory	Mary L	0.50	Passed on retest
2. Departments & Routings	2.1	None	Define Resources	Bill of Mat	Jane D., John P.	0.50	Passed
	2.2	2.1	Define departments and assign resources to departments	Bill of Mat	Jane D., John P.	0.20	Passed
…..	…	…	……	….	…….	…..	…….

due to the level of knowledge about the product (the developers can see "inside the black box"). In contrast, packaged software from commercial vendors is typically introduced "as is." The patch or upgrade in its entirety is moved into production; incremental modifications are not possible. If the change fails, it must be completely backed out and the previous version restored.

As a consequence, the emphasis for changes to purchased software is relatively more towards integration and regression testing. While vendor release notes are useful, there is some level of uncertainty that demands thorough testing across all affected modules. In addition, strong library/release controls are necessary to ensure backouts are performed correctly when needed. Of course, in-house changes sometimes fail in production and have to be backed out as well. Ultimately, it is the complexity of the change, including the number of modules/systems affected, that drives the risk and concomitant level of testing/review.

Urgent/Emergency Changes

Assume an organization's ERP stops during business hours. Possibly the underlying Oracle database has become corrupted or a key server has mysteriously lost both the primary and backup power supply. Regardless of the reason, a "hard down*" condition demands a high-speed fix. The question then becomes, does a three-alarm emergency justify making changes without testing and with only a single approver?

* Sometimes called "severity one."

The answer depends on the nature of the fix. If a server fails and is replaced by an *identical* unit, little testing may be required, assuming that all files and the operating system have been successfully restored from backup media. Applications, on the other hand, require testing, even under emergency conditions. The testing may not be extensive, but raw code thrown into production without a test is unacceptable.

If an emergency change is successful, the modified system is obviously now in production status. Assume the change is significant. Under the normal process, not emergency, a user acceptance test, integration, and/or regression test may be required, along with formal signoff. Should a hastily implemented change, now in production, be retroactively tested? The hard-nosed answer is yes—the changes may result in errors not yet detected. On the other hand, production is a strong, albeit dangerous, testing ground. Some organizations compromise by requiring IT management and responsible users to document reasons for retroactively testing or not testing. IT organizations must rationalize testing dollars.

Integration of Code (Micro) and Enterprise (Macro) Level Change Control Systems

A robust enterprise change management system provides end-to-end accountability. A change, such as the addition of a new ERP function, should be supported by an audit trail of all its constituent code or script changes. For example, assume a payroll system is modified to account for pay based on a new piecework formula. An entry in the macro level change control system (e.g., NetResult's Problem Tracker) should contain a detailed description of the change, testing by both users and IT (unit, integration, regression, etc.), approvals to start the work, and authorization to move it into production. The macro entry (e.g., payroll change #12309) should also include a link to a lower level (micro) change management/versioning system, such as Serena Corporation's PVCS. The micro change management system will document the change control number found in the macro level system.

As a result, changes can be traced both backward and forward. Without a tie in, there is no assurance that changes at the code, script, object, or other executable level are properly authorized. In other words, an auditor could determine, via length/date comparison, that module xyz.exe changed from one month to the next. Without the embedded links to the specific authorization, however, it is not clear which approval serves as the approval for the change.

SELECTING THE RIGHT CHANGE MANAGEMENT SYSTEM

Ideally change management is integrated with an organization's SDLC, problem management/incident reporting, directory services (LDAP), and authorization infrastructure. With the integration of all these control elements, the degree of control over IT changes increases disproportionately. All parties find the system easier to use and compliance is more likely than with stand alone systems requiring duplicate entry. It is important to note that although the SDLC should be linked to and integrated with change management, the two processes are separate governance systems. To be effective, change management should not assume too many duties. It asks whether all the work performed relative to a change or project is adequate to allow a move into production. It is not a comprehensive best practices enforcement system.

Versioning

There are many development environments on the market such as Eclipse, IBM Websphere Studio, SAP NetWeaver Developer Studio, and Visual Studio.net. Within these environments, code, scripts, database schema changes, and documentation need robust version controls providing the following minimum functions:

- Check in/check out;
- Visual differencing (highlight changes between versions down to the line of code or script variance);
- Versioning for schemas, scripts, JCL, documentation, and other system elements; and
- Audit trail of changes.

TABLE 4 Change Management Segregation of Duties Model

Assigned duty	Requestor	Approver	Developer	Tester (from user perspective)	Production move
Initiate change requests (requestor)	√	x	x	x	x
Authorizes and approves change requests (approver)	x	√	x	x	x
Makes changes in development environment	x	x	√	x	x
Tests the change (from end user perspective)	√	√	x	√	x
Moves change into production	x	x	x	x	√

Note: "X" indicates duties cannot overlap (e.g., same individual cannot perform both functions)

Authentication and Workflow

Few would regard hard copy-based signatures as an efficient means to document change authorizations. Nonetheless, a surprising number of organizations continue to use manual signatures on paper as evidence of management approval of changes. Electronic authentication, on the other hand, provides many benefits beyond the elimination of inefficient manual processes. A change management system that integrates with LDAP* taps into the organization's existing security infrastructure. Using current names and email addresses, notifications of pending and completed changes may be sent as the change request flows through the system. Finally, segregation of duties can be enforced by setting up tables relating specific change control authorities to organizational levels and specific individuals.

SEGREGATION OF DUTIES

Maintaining proper segregation of duties in information technology is an ever-present hurdle. For example, a small IT group may have only one network administrator. Who reviews network changes? The answer has to be a second party, even if that individual is not a technical peer. Imperfect segregation of duties is better than none, even if the reviewer is a peer developer. The practical inability to achieve an ideal change control environment does not justify the omission of a workaround. A strong change management system prohibits changes that have not been reviewed by at least one person other than the individual doing the work. Table 4 shows the most commonly accepted segregation of duties model for change management.

MECHANICS AND MIND-SET

Most developers learn programming basics either formally, in technology classes, or informally via on-the-job training. In any case, most developers' first products are likely to be *private* programs. In other words, the same person writes the code from the bottom up, tests the program, and perhaps documents it. However, systems written for an organization's business/production environment are *public* programs, used by multiple individuals, documented, tested, potentially reusable and part of larger units. Public† programs do not belong to the developer. Hence, there is a mind-set change required to mitigate the tendency towards a private program perspective. Change control, documentation, and other standard controls practices need continual promotion and support, otherwise systems devolve into a mere concatenation of private programs and system fiefdoms.

To ensure developers, users, and management participate in change control and avoid the tendency to privatize, the mechanics of the process should be straightforward, clear to all parties, and supported by management. Good practices include the following:

* LDAP (lightweight directory application protocol) is an open and configurable protocol; a wide variety of information about an organization, including security, can be housed within its structure.

† "Public" here is defined as belonging to the organization that ultimately pays for the system (no relation to "public domain").

- *Weekly meetings* set up specifically for discussion and approval/disapproval of changes. Participants from multiple IT groups may identify potential conflicts.
- *Schedules and lead times* for approvals, testing, and production moves. For example, managers cannot be expected to review complex changes one hour before the weekly meeting. Typically a schedule of lead times related to change size is published. Change control procedures may require one, two, and five days for "lite," medium, and large changes, respectively.
- *Informal presentations* for medium-high risk changes. Peer review offers many benefits including the avoidance of scheduling conflicts and identification of problems by those not directly participating in the change implementation.

CHANGE MANAGEMENT SPECIALIST

For a large organization, a change management specialist provides the practical support necessary to keep the policies and procedures working at the day-to-day level. Specialists review submissions, report on procedure failures, provide management with high level reporting (e.g., percentage of back-outs by month), train new participants, and evaluate the quality of documentation. Specialists can also perform ad hoc testing to monitor compliance.

Time pressures may tempt developers, users, and other change management participants to skip or skimp on required forms, even if the functionality is automated. The specialist can offload some of that work and rationalize the system over time.

SUMMARY

At first glance, the full change management process appears bureaucratic and complex. However, it can be made to work efficiently if the tasks are batched and the participants meet regularly (a weekly change management working session is typical). Templates, workflow software, and electronic signatures smooth the process.

Admittedly change management is hard to do well. Some organizations have made top-level decisions to implement change control without understanding the consequences. If, for example, the policy requires integration testing for medium-high risk changes and the IT staff has never *seen* an integration test template, then the implementation will be flawed. But with electronic tools, an incremental approach and management support, a sustainable and auditable change management system can become a vital component of the IT general control infrastructure. And, of course, it is essential for good IT governance.

ACKNOWLEDGEMENT

Special thanks to Carolyn Treen, information technology consultant, in Boston, Mass., for her contributions to this article.