Summary	Definition
No command line access	No command line access to the operating system of any server on which the system runs shall be granted to anyone involved in the day-to-day operation of the system. This includes anything equivalent to command line access, such as uncontrolled applications and the copying in of arbitrary programs, scripts, or other files. The motivation for this requirement is to force every action needed to operate the system to be available via a function whose use can be controlled and recorded, and to minimize the number of people who need command line access. This requirement does not apply to steps necessary to rectify serious system problems, or to install, upgrade, or reconfigure the system. (Nevertheless, in those cases, this requirement should be relaxed only as far as is necessary to get those jobs done).

Extra Requirements

None, beyond those common to both types of user authorization requirement.

Considerations for Development

Specific authorization requirements can be varied, and the ways to implement them can be equally varied. Deal with each one on its merits. Don't seek a grand mechanism that can solve them all: they might be too diverse.

The biggest decision is whether configurable access control is worth implementing, even if there's no explicit requirement for it. You can't expect a requirements specification to specify every little detail of who's allowed to access what. Judge for yourself how big the gaps are. Maybe there's an unspoken expectation of configurable access control; if so, bring it out into the open and resolve it quickly.

Considerations for Testing

Testing a specific authorization requirement is reasonably straightforward: can that kind of user access whatever it refers to? Are all other kinds of users prevented from accessing it? If a user is able to delegate their authority to another user, test that a delegatee is authorized whenever the delegator is. Change the kind of a user (if that's possible, for example by assigning them a different role), and test that their authorizations change accordingly.

An invaluable tool when testing authorizations is for all rejected authorization requests to be chronicled (logged). When reviewing requirements, check that this feature is included. If not, insist on it.

11.5 Configurable Authorization Requirement Pattern

Basic Details

Related patterns:	Extends user authorization; refers to authorization access, chronicle
Anticipated frequency:	Up to two requirements
Pattern classifications:	Affects database: Yes

Applicability

Use the configurable authorization requirement pattern to specify that the definition of which users can do what is to be configurable (that is, can be changed dynamically).

Do not use the configurable authorization requirement pattern to specify what a set of users is authorized to do or see; use the specific authorization requirement pattern for that.

Discussion

Configurable authorization control is wonderful: it lets you change who can do what, whenever you like. An employee needs more authority to do their job properly? No problem! A company restructure? Bring it on! At the same time, it's a millstone, with major drawbacks: it's difficult and expensive to implement; it can incur performance penalties; it's time-consuming to set up and subsequently to maintain; ill-considered (or malicious) changes can have major consequences; and it introduces security risks of its own. So ask for access control to be configurable only if it's worth it; even then, keep it as simple as possible and use it only in those areas that justify it. (That is, don't use it everywhere just because it's there.) Also, take care to make clear what's needed and what isn't, so you don't inadvertently end up with an over-fancy solution.

Configurable authorization is most fruitful when used to control access to *functions*, because in any serious system, the functions are too numerous to fix their access correctly up front, for all time. Configurable authorization to *data* is rarely called for, because a normal system simply doesn't need the ability to partition data in arbitrary ways; the cases that do occur are usually isolated and best dealt with using specific authorization requirements. (Note that even without requirements for configurable authorization to data, the system might still be implemented with access control at the database level, on tables and SQL procedures and so on. That doesn't concern us here.) Sometimes, though, several sets of access configuration to functions are needed—one for each of a set of data values. For example, a multi-company system might allow permissions to be defined differently for each company within it—such that one user can access one collection of functions for one company and other functions for a different company. This renders configurable authorization more complex to implement and to manage, so avoid it if you can.

Give configurable authorization requirements a high priority, because if you need access control, you can't very well live without it for an extended period until it's delivered in a later phase—and any attempt at an interim solution could be a mess. In any case, its priority must be at least as high as any specific authorization requirements that depend on configurability.

Content

A configurable authorization requirement should contain:

1. **Class of users** For which users do we want to make access configurable? Do it only for classes of users that need flexibility. Avoid it for classes of users that have largely fixed privileges—which, in most systems, includes *customers*. If possible, avoid it for high-volume classes of users (again, customers)—because of the likely performance impact.
2. **Nature of access** Are we asking for configurable access to functions, or to data, or to a combination of both? Describe what's needed in as much detail as you can. Make clear the significance of what you're asking for; don't let a massive undertaking look innocuous.
3. **Motivation** Why do we want this to be configurable?

Template(s)

Summary	Definition
«*User class*» access to «*Nature of access*»	A «*User class*» shall be able to access only «*Nature of access description*» to which they have been granted permission [(by virtue of the roles assigned to them)]. The motivation for this requirement is «*Motivation statement*».

Example(s)

Summary	Definition
Employee access to functions	An employee shall be able to access only those internal functions in the system to which they have been granted permission (by virtue of the roles assigned to them). An internal function, for the purpose of this requirement, is one that is not intended to be available to customers. Within each function, the ability to perform each distinct type of action (view, create, modify, delete, approve, and so on) shall also be restricted in the same manner. The motivation for this requirement is to allow the system to adapt to the skills of the workforce that uses it and to be able to react to reorganization of the company.
Employee access to data	An employee shall be able to access only those classes of data to which they have been granted permission (by virtue of the roles assigned to them). For the purpose of this requirement, "class of data" means any body of data that can be defined precisely such that no item of data shall belong to more than one body of data. This is not to say that explicit access must be granted to every item of data; it means control can be given for specific things that warrant it. A user shall have no access to data that is subject to such control, unless permission is explicitly granted to them. (That is, a **denial by default** approach shall be taken.) The motivation for this requirement is to allow data to be segregated in various ways so that a range of powerful inquiries can be made available to any user while still permitting them to view only the data relevant to them.
Employee access to functions per company	An employee shall be able to access only those internal functions for a particular company to which they have been granted permission (by virtue of the roles assigned to them) for that company. "Internal function" shall be taken to have the same meaning as in the requirement before last. The motivation for this requirement is to allow the operation of each company to be defined separately, while still permitting employees to do work for more than one company.

The first two examples spell out the two main "natures" of access control (functions and data). A more common way to express these sentiments might be, "A user shall be able to access only those functions and that information to which they have been granted permission," but that fails to address what information it might apply to, and it hides the enormity of what it's asking for to such an extent that only an eagle-eyed reader would spot it.

Extra Requirements

A configurable authorization requirement can have the following kinds of extra requirements, in addition to those common to both types of user authorization requirement (each of which is discussed in its own subsection that follows):

1. **Defining user roles** We need a way to set up user roles and define what privileges each role has.
2. **Authorization inquiries** Access configuration is hard enough to manage at the best of times, so provide inquiries to make it as easy to understand as possible: what each person is allowed to do, which people have each privilege, and so on.
3. **Chronicle authorization changes** Keep track of changes to any of the configuration that controls who can do what.

Defining User Roles It's most convenient to discuss roles in a reasonably abstract way (because it avoids repetition), which makes sense only if we assume that a user can assume several such roles. A role, then, doesn't necessarily equate to a job description–nor to an "actor" or "class of user." (Roles are likely to be more numerous than any of these other things.) Even if roles are to be configurable, it's useful for the requirements specification to list candidate roles informally; it gives readers a better idea of what we're talking about and can give the configuration a head start.

We need to specify a tool for defining user roles:

Summary	Definition
User role maintenance	There shall be a function for the creation and editing of user roles. Each role shall have a name and a description of its purpose. It shall be possible to maintain a list of the access privileges granted to any user with this role.

If we're defining roles, we could be picky and make sure the software actually uses the roles when deciding what a user is allowed to do:

Summary	Definition
Access control per role	The privileges a user has shall be determined by the roles to which the user is assigned. Privileges shall not be given directly to each user.

Authorization Inquiries Why systems act as they do is always shrouded in mystery–invariably far more than necessary. The workings of serious access control can be equally mysterious if it doesn't let you see what's going on. A set of inquiries for this purpose is invaluable. The two most important ways in which authorizations should be viewable are **by user** (to see what each user can do) and **by privilege** (to see everyone who can access each thing). The latter can also point out

anything to which *no one* has access, which means parts of the system are lying idle. Here are a couple of basic examples:

Summary	Definition
User authorization inquiry	There shall be an inquiry that lists all the access privileges granted to a selected user. For each privilege, it shall indicate which of the user's role grants it. Every user (except customers) shall be able to inquire on *their own* access privileges.
Privilege types inquiry	There shall be an inquiry that lists all the types of access privilege known to the system. For each privilege, it shall name all the user roles that grant it (and, in so doing, identify privileges granted to no one).

The second example leaves open the question of where the information about privileges "known to the system" comes from and how it gets there. The most obvious answer is that it's stored somewhere and there's a way to edit it. If you want to make that explicit, write a requirement to that effect; something like this:

Summary	Definition
Privilege type maintenance	There shall be a function for the creation and editing of information about access privileges. Each privilege shall have a name and a description of its purpose.

Chronicling Authorization Changes Changes to the configuration of access control can have major consequences, so insist that all changes to access control are chronicled (as per the chronicle requirement pattern in Chapter 7). If a malicious user granted their own access to an extra function, used it, and then revoked access, chronicle entries could be essential in piecing together what happened. Refer to the chronicle requirement pattern for more.

Summary	Definition
Record changes to access control	Every change that affects the access privileges of any user in any way shall be recorded, including at least: ■ Date and time. ■ User ID of the person who made the change. ■ Full details of the change.
Record authorization check	Each check of whether a user is authorized to perform a particular function or action shall be recorded. This record shall include the *context* of the check, which means that if several functions invoke the same subfunction, the record shall indicate which function invoked the subfunction. It shall be possible to turn the recording of authorization checks on and off. (It is normally unnecessary to record checks in a production environment, but it's useful when performing certain kinds of testing on the system and when diagnosing problems.)

Considerations for Development

If access control logic is simple and fixed, you can hard-code it. If it's not simple, or if it's liable to change, it's better to build a general, configurable mechanism–regardless of whether the requirements say you should. However, developing an underlying mechanism for access control is difficult and not for the faint-hearted. Apparently sensible and logical approaches can turn out to be a nightmare to manage in practice, because of the sheer quantity of configuration data. If you think a general access control mechanism will be worthwhile, bite the bullet early. Trying to retrofit it later is likely to be unpleasant.

Considerations for Testing

First of all, recognize that you're testing the requirements, not the privileges of whatever roles might have been set up: testing isn't responsible for checking that every user can do what they need to do their job. Having said that, it's always sensible to configure a test system as closely as possible to the live environment–which means defining roles and their privileges as realistically as possible.

When testing access to functions, for each function, test two things: first, that access to the function is controlled; and second, that the correct privilege is checked. This is exceedingly tedious to do manually. It helps if the system lets you observe each authorization check (down to the name of the privilege being checked); then you can see directly whether the correct privilege is used for each function. One way to observe authorization checks is to record them in a chronicle, and there is an example requirement for doing this in the "Chronicling Authorization Changes" subsection of the "Extra Requirements" section in this pattern.

A simpler test is to create a user role that can do everything, and verify that a user with that role can indeed do everything. Also create a user role that can do nothing, and verify that a user with that role can do nothing but publicly-accessible things.

One difficulty is that the number of privileges could keep growing. You'd have to be inhumanly diligent to insist on trying to test every new one that comes along.

11.6 Approval Requirement Pattern

Basic Details

Related patterns:	None
Anticipated frequency:	Often no requirements; rarely more than half a dozen requirements
Pattern classifications:	Functional: Yes; Affects database: Yes

Applicability

Use the approval requirement pattern to specify that a particular action (or set of actions) must be approved (or, in *some circumstances* approved) by a second person before it takes place.

Discussion

Any function in a system can be constructed so as to require approval before it takes effect. We usually think of one person approving an action requested by another, but it is also possible to allow a person to approve actions suggested by the system itself. ("An upgrade is available for the Whizzo Happiness Calculator. Would you like to download it now?" is an example.)