## *BP.2 Develop Stakeholder Requirements*

The identification of clear, complete and unambiguous requirements is one of the more difficult activities to achieve successfully, yet contributes most significantly to the overall success of the development.

In aiming for success, a good starting point is to establish who the ultimate customer is and all the potential stakeholders. Once the stakeholders have been identified and their roles clarified, the stakeholder requirements can start to be formally captured. The starting point will usually be through some form of initial customer correspondence, possibly a request for a quote, a formal order or purchase order, or through a tender specification as part of a formal tendering process. Internally, this correspondence could be an instruction from the executive team or a specification from the marketing department.

A formal review of the initial correspondence provides a good starting point for establishing the full set of stakeholder requirements. Indeed, it may be all that is necessary. However, in many cases, further work on capturing the full requirements will be needed, and the review of the initial correspondence can help identify subsequent actions that need to be taken. It may also identify additional stakeholders that were not initially considered.

Typically, the subsequent actions necessary involve gaining a better understanding of what is required, why it is needed and what business objectives are being addressed by the customer. This type of information helps greatly in developing a product that satisfies its intended purpose and expectations, rather than simply being built to order. Again, there are many approaches to this, including discussions, workshops, demonstrations, prototyping and modelling, but whichever method is used the final results should be formally documented and reviewed (see TEC.4 Verification).

Even after taking all appropriate actions, in some cases it might not be possible to achieve the level of understanding that is desirable (e.g. where the customer acknowledges that they are not fully clear on what they require). This does not mean that the exercise has failed, but, in fact, can actually help greatly in ensuring a more successful product by allowing a more appropriate lifecycle model to be selected for this type of situation. Conversely, care should be taken to watch out for the situation where it is believed that the stakeholder requirements are fully complete but, in practice, deficiencies exist. This is more likely to be the case when not all the stakeholders or inappropriate stakeholders have been identified or involved.

Even if a good set of requirements has been established initially, it is very important to continually monitor the requirements as they are developed. This is to ensure that any identified gaps or ambiguities are addressed as soon as possible, and that the development does not suffer from uncontrolled requirements (or scope) creep.

The stakeholder requirement definition, in whatever form it takes, should be managed under the configuration and change management system. This is particularly important for the set of requirements, as it ensures that appropriate baselines are identified from which all further development activities will follow.

One additional aspect should be implemented at this stage to help towards a successful development. This involves ensuring that the requirements can be traced throughout the chosen lifecycle to the final validation activity. Requirement traceability can be achieved in a number of ways, ranging from the use of specific requirement management tools, to the careful structuring and numbering of the requirements. Large textual blocks are not conducive to enabling such traceability, and such situations should be avoided.

## *BP.3 Validate Stakeholder Requirements*

As part of the process, the stakeholder requirements should be validated with the customer to ensure that they continue to fulfil the customer's needs and objectives. In some situations this will be occurring in parallel with establishing the requirements through discussions, workshops, etc. that involve the customer. In other cases it may be necessary to submit the documented stakeholder requirements to the customer for review and approval. In either case, but more importantly in the latter case, care should be taken to ensure that the stakeholder requirements are expressed in such a way that the customer can reasonably be expected to understand what is being presented.

As part of the validation activity, any issues or concerns should be addressed before final agreement is sought from all stakeholders identified to approve the requirements. Even if the stakeholders have been actively involved in preparing the requirements, there should be some formal point in time where the requirements are approved for use and placed under formal configuration and change management. Any unknowns or concerns at this point should be considered as risks and tracked through risk management.

It should be noted that, while this is an initial validation activity aimed at mitigating risks to the development process, the final validation of the requirements cannot be considered


 

complete until the product is available for test in its intended environment. Nevertheless, effective validation here will contribute greatly to a more successful development activity.

With some lifecycle models, validation activities actually continue through the lifecycle phases, especially when the Stakeholder Requirements Definition process has identified that the requirements are not as complete as needed. Even in these cases, formal requirement validation should still be undertaken to ensure that what does exist is correct – so that subsequent development is not misdirected.

## *BP.4 Manage Changes to Stakeholder Requirements*

The final base practice addresses changes to the baselined requirements that might occur throughout the development process. Again, this is an area where many organizations have difficulties. Changes have been uncontrolled, and the development has become impossible to manage to the agreed schedule or budget through the resulting creep in requirements.

The prerequisite for handling changes effectively is that the stakeholder requirements are under appropriate configuration and change management. The need for identified changes can then be managed through the formal change management process.

In some cases, organizations set up a formal change control board that involves the customer and appropriate stakeholders. This approach can address all the requirements set out in this base practice, and may involve a formal stakeholder meeting held on a periodic basis, a virtual change control board using electronic tools, etc. Irrespective of how the changes are managed, it is important that appropriate levels of stakeholder representation are involved in the process. In many cases, where stakeholder requirements need to be changed there will invariably be an impact on the schedule, budget and functionality. For example, where the customer is requesting a reduction in the budget or schedule, the effect will probably be on what can be developed in the time available. Here, the functionality may have to be reduced or be phased, or the cost may have to increase because extra resources are needed to maintain the development in a shorter timescale. Similarly, if additional functionality is required, this will almost certainly affect either the budget or the schedule, or both. Therefore, participants in the stakeholder change process should have appropriate authority to make the necessary decisions and agreements.

In many cases, not all the stakeholders will need to participate or, for various reasons, will not be able to participate, in the formal change process. Therefore, it is essential that

following the change process all necessary stakeholders are made aware of the result and the implication of the changes that have been approved, and in some cases also of those that have been rejected.

## Requirements Analysis

There is a close link between the Requirements Analysis process and the Stakeholder Requirements Definition process, which precedes it, and the Architecture Design process, which follows it. However, these processes are rarely truly sequential, and, in many cases, there will be a fair degree of overlap and iteration through the initial stages of a project.

The output of the Stakeholder Requirements Definition process is typically customer focused around the functionality, objectives and needs of the product development, whereas the input to the Architectural Design needs to be system or engineering focused. Therefore, the purpose of the Requirements Analysis process is to transform the stakeholder requirements into a set of structured system requirements that form the basis of the product development.

Clearly, the amount of requirements analysis that will be needed is dependent on many factors, but is influenced mostly by the nature of the stakeholder requirements. In some situations the stakeholder requirements may already be sufficiently engineering orientated that little additional analysis is required. For example, in very large development projects, where multiple suppliers are involved in developing subsystems, the prime supplier may well provide subcontract suppliers with very detailed engineering requirements. Similarly, in very small enhancement projects, the customer requirements may be sufficiently straightforward and understood by the supplier that no, or very little, further analysis is necessary.

In other situations, the stakeholder requirements may be very high level and focused virtually entirely on the business or operational needs of the customer. For example, the prime supplier in the example above may well have received an operationally specified requirement for the product, such as is often seen in the public sector. Even in small developments or enhancements, the stakeholder requirements can appear system focused on the surface, but on further consideration may not actually be what the customer really needs.

Irrespective of how much requirements analysis is necessary, as a minimum the stakeholder requirements need to be evaluated in terms of their suitability for continuing with the


 

architecture design. It is at this point that the development organization will get its first real idea of the size of the development project. Product sizing is probably the most important attribute of any development project, as it:

- provides one of the key inputs to the estimating process
- offers a unique mechanism for monitoring potential requirement creep
- enables the ability to normalize data over multiple projects
- forms the basis of many high maturity practices.

As with many other process outcomes in TickIT*plus*, the outcome for the Requirements Analysis process aims at a high standard, but in practice the limitations do need to be recognized. The outcome clearly expects a set of system requirements to exist, but also that they are produced in such a way that there is no subsequent unnecessary rework to them. Reworking the system requirements can happen for many valid reasons, such as following approved changes to the stakeholder requirement, but every effort should be taken to ensure that errors and mistakes later in the development are not attributed back to a weak Requirements Analysis process.

There are four base practices that make up the BPL Requirements Analysis process. These are:

BP.1 Develop System Requirements
BP.2 Estimate System Requirements Size
BP.3 Manage System Requirements
BP.4 Manage Changes to Requirements.

## *BP.1 Develop System Requirements*

As indicated, the amount of requirements analysis and when it is performed depends largely on the process for defining the stakeholder requirements and the nature of the resulting output. If stakeholder requirements are very high level and focused virtually entirely on the business or operational needs of the customer, the requirements analysis activity could be quite extensive.

There are a number of different approaches to performing a satisfactory requirements analysis activity, but in the main they all tend to rely on the use of appropriately skilled resources and strong review techniques.

If not already done during the Stakeholder Requirements Definition process, due consideration should be given to any internal developments and any underlying core product development strategies. For example, in analysing the stakeholder requirements some system requirements could be identified that may need to take into account certain organizational constraints or goals, for example relating to performance. This typically is the case when the customer development involves customizing a core product.

Finally, the system requirements are formally reviewed (see TEC.4 Verification), and placed under formal configuration management (see PRJ.3 Configuration and Change Management).

## *BP.2 Estimate System Requirements Size*

At some point during the Requirements Analysis process, typically towards the end or, alternatively, throughout, an estimate of the product size should be generated, reviewed and documented. The size estimate is not given in hours or effort, as these would typically result from the size estimates. Size can be expressed in many ways, ranging from the classic lines of code and function points, to use cases, interfaces or simple requirement counts. The important aspect here is that the approach to sizing does not have to be scientifically accurate, as it is only intended to provide additional estimating information.

Size estimates provide or support the estimates of cost, scheduling and resourcing. Given a common size parameter that is used across projects, and the collection of actual project performance data such as costs, estimating can be based as much on historical data as on the specifics of the individual project. While this practice is covered in the Requirements Analysis process, which is most often undertaken by the engineering discipline, the estimating of costs, schedules amid resources is more commonly performed by the project management group. However, TickIT*plus* base practices do not infer any particular order or any particular implementation group.

## *BP.3 Manage System Requirements*

An important characteristic of the system requirements is that they are structured to show the sequencing, prioritization, functional, non-functional and interface requirements. In addition to driving the architectural design, the system requirements will form the key input to the system testing activities.


 

In a similar way to the stakeholder requirements, the system requirements should be identified in such a way as to facilitate traceability back to the stakeholder requirements, and forward through the development, integration acceptance and final release.

As well as placing system requirements under configuration management, agreed requirements should be formally baselined. In essence, the distinction is that placing the requirements under configuration management will ensure that changes are formally undertaken, whereas creating a formal baseline at a specific point in time provides a snapshot of the requirement set. In many cases where the requirements are textually based in a single requirement specification, these will occur naturally together but, if requirement management tools are used, this may not always be the case. For example, each requirement could be individually version controlled, or there could be multiple versioned requirement specifications that collectively make up the requirements baseline.

### *BP.4 Manage Changes to Requirements*

The final base practice, as with many of the TickIT*plus* processes, addresses the management of change. The prerequisite for handling changes effectively is that the system requirements are under appropriate configuration and change management. Identified changes can then be managed through the defined Configuration and Change Management process.

Whereas changes to the stakeholder requirements more often originate from the stakeholders, changes to the system requirements can originate from either change to the stakeholder requirements or internally from the organization or development project. Some internally required changes can be managed completely internally, while others may need to go through the same mechanism as for changes to the stakeholder requirements. In either case, it is essential that, following the change process, all necessary stakeholders are made aware of the result and implication of the changes that have been approved or rejected.

## Architectural Design

The purpose of the Architectural Design process is to produce a top-level design that identifies the major components and interfaces of the product. The word ‘product’ here may be taken to mean a complete product, a product component, a service or a service component.

Architectural design usually occurs on a product development project or programme of change within the organization. Depending on the lifecycle model used, it can be undertaken as a separate stage or as an inherently iterative process working intimately with the Development Implementation process. In the latter case, it is usual for both to be repeated, as the solution is developed in the best way to meet the system requirements. A top-level design may need to be modified after one or more of the components have been designed.

Many stakeholders are often involved in the Architectural Design process. However, there is usually a single technical authority on the project (e.g. a senior designer), who has the responsibility to ensure that the system design is adequately performed, and that the system requirements are met. Where there are multiple internal groups involved in a project, the project will usually establish appropriate controls to co-ordinate intergroup issues. This is often achieved through a group consisting of technical team leaders, from the disciplines involved, who have the responsibility for analysing and accepting the system requirements.

The successful completion of the Architectural Design process usually results in a top-level or system design for the product, which has been reviewed, approved and maintained under controlled conditions (see PRJ.3 Configuration and Change Management).

As with other lifecycle processes identified within TickIT*plus*, the outcome of the Architectural Design process states that there should be zero defects identified in the subsequent phase that were introduced as a result of architectural design activities. However, in practice this would rarely be seen, especially given the complex nature of products being developed these days, but nevertheless the organization should be proactive in trying to reduce such defects.

In order to aim for zero architectural design defect leakage, the organization should establish clear verification criteria for designing and reviewing top-level designs. In addition, it should capture and record all downstream defects found, analyse them for root causes and put in place actions to remove them from the Architectural Design process.

There are four base practices defined in the BPL Architectural Design process that work collectively to achieve the process outcome. These are:

BP.1 Establish Development Approach
BP.2 Create Architectural Design
BP.3 Review Architectural Design
BP.4 Manage Architecture Changes.


 

## *BP.1 Establish Development Approach*

The starting point for establishing the development approach will normally be through the selected development lifecycle chosen. However, this will only provide a standard organization lifecycle, and it is therefore important to consider it in light of the particular customer needs and development requirements.

Make, buy or reuse analysis begins early in the project, when requirements are being developed, continues during the system design, when different design options are being considered (at both high and detailed levels), and completes with the decision(s) to make, buy or reuse products or product components. The decision whether to make, buy or reuse is usually a trade-off between technical requirements and cost/time considerations.

Bought-in items are rarely a zero-effort solution for the project. If the functionality is not exactly what is required, or the interface works differently, then effort will be required to deal with this and possibly constraints will be imposed on the system design. These factors, and the effort and cost they impose on the project, should be taken into account when considering the approach to be taken.

Also to be considered is the cost and effort required to maintain the bought-in item over the lifetime of the product or product family. This may require the production of a buy-in strategy detailing who will support the item and the timescales and responsibilities for support.

Factors affecting the make, buy or reuse decision should include the following:

- the functions the products or services will provide and how these functions will fit into the project objectives
- available project resources and skills, and the impact on core organizational competencies
- the costs of acquiring versus developing internally
- critical delivery and integration dates
- strategic business alliances, including high-level business requirements
- market research of available products, including COTS products
- the functionality and quality of available products
- the skills and capabilities of potential suppliers
- licences, warranties, responsibilities and limitations associated with products being acquired
- product availability

- proprietary issues
- risk reduction.

The selection decision and supporting rationale should be documented, reviewed and approved. Where the decision is strategic or complex, decision grids can be used to help guide the decision.

## *BP.2 Create Architectural Design*

The system requirements are analysed to identify the major system components (e.g. logical groupings, services). In doing this the analysis should note any interface requirements between the major components and the external environment, along with any design constraints (e.g. interface standards, communications protocols) that may arise. The analysis should also consider key non-functional requirements, such as the need for reuse, information policies, security arrangements and performance requirements.

The design should also consider the target operational environment that is likely to be in use at the time of product release. Changes to the operational environment typically have a long response time due to the sensitivity and complexity of most operational environments. If any changes are to be made, then the sooner they are initiated, the better.

Identifying clear design criteria that are used to improve the quality of the design also reduces the time taken to produce the design and ensures better efficiency of the review. Often these criteria will be expressed as internal design standards. An example of such criteria is that modules should exhibit internal integrity and minimal external coupling. If two components are identified with a large number of interactions between them, this would be a good indication that they should be grouped, or that the functionality should be split along different lines. Conversely, if the system design has little (but key) interaction between a minimum number of components, this can be taken as an indication of good system design and a good (stable and robust) architecture. In general, components should be grouped by their interactions rather according to any preconceived ideas about how the functionality should be split.

As the design becomes clear, the requirements can be mapped onto design elements to ensure that all system requirements are addressed by the design. Where system requirements are split between more than one component, there will be a need to note


 

the dual mapping of the requirement, or to develop derived requirements showing which part of the requirement each component is responsible for.

The output of the Architectural Design process should be descriptions of a set of components that can be developed in relative isolation from the other components, and of the interactions between them. Techniques such as 'use cases' or 'operational scenarios' are helpful in clarifying the architectural design and providing consistency in its communication.

## *BP.3 Review Architectural Design*

The purpose of reviewing the top-level design is to identify and remove any defects that might have occurred during its development. Defects cost dearly – costs incurred in making them in the first place, cost involved in trying to find them, costs to remove them once found and, not forgetting, the opportunity costs of what could be being done with the extra effort being expended. In addition, the greater the gap between the introduction and the removal of the defect, the more it costs to remove it. It is no surprise that defect removal is one of the primary quality improvement strategies in use today. Ensuring the top-level design is defect-free is one of the most powerful ways of ensuring that a project meets its cost and time forecasts.

The effectiveness of reviews is raised massively by the involvement of independent experienced staff who can provide a 'fresh' view on the design. Free of the developing mindset of the design architects, and unencumbered by the restrictions of the project timescale, they will often find defects that the author or author team cannot. If the purpose of reviewing is to find defects, review by one or more experienced peers is the most effective way of doing it.

The efficiency of the review process is raised by having a standard process, with trained and skilled people who know the process well and have used it many times. In particular, having a review leader who keeps the review moving forward and prevents any attempt to solve the defects immediately greatly shortens the time required for the review. The defects should simply be identified in the review, and solved later.

By selecting other stakeholders in the development, such as those from support and maintenance, to act as peer reviewers, the reviews can also contribute to efficient two-way communication. The peer reviewers learn about the system being designed. The author team learns generally from the experience of the peer reviewers, and especially about

the sort of mistakes that they make. This builds their ability to avoid such mistakes in the future, and provides the peer reviewers with an excellent early insight into the product and requirements.

The formality of the review process is best adjusted to the circumstances of the project or organizational area. In all cases, however, it is the rigour of the review process that determines its effectiveness in identifying defects. The effort in defining, implementing and maintaining a rigorous review process is more than repaid by the effort saved in fixing defects, particularly those found late in the product lifecycle (see TEC.4 Verification).

### *BP.4 Manage Architecture Changes*

Changes are a fact of life, and in some lifecycles are actively encouraged. Changes can come from outside the organization (e.g. a change in legal requirements), outside the project or organizational area (e.g. a change in business requirements) or from within the project or organizational area (e.g. from discovering a defect).

Once the system requirements have been approved and work has started on the top-level design, changes to any of the key configuration items (e.g. customer requirements, system requirements, code) may occur. Such changes must be carefully managed. The mechanism for this may vary from a discussion and agreement as part of an agile team, for example, to a formal process involving multiple stakeholders.

## Development Implementation

The purpose of the Development Implementation process is to transform the system requirements and architectural design into a product. The word ‘product’ here may be taken to mean a complete product, a product component, a service or a service component.

It is taken as axiomatic that there are requirements of some sort, although often these may not be of sufficient detail or accuracy to create a product. If the requirements are felt to be inadequate, it is better to spend time creating a suitable set of requirements than hoping to just ‘get through it’. However, it is possible to illuminate areas where the requirements are deficient by moving forward with the design. In the case of, for example, an agile team, or a team with regular and easy access to a customer or customer representative, this can be a most efficient way of working. If this approach is taken, care must be exercised to


 

ensure the integrity of the system requirements with the design. A mechanism such as a traceability matrix is very helpful in such situations, although requirements traceability is always useful.

What makes a set of requirements suitable will depend on the product being created. Typically, the set includes not just the functional and non-functional requirements, but also some measure of the quality of these requirements. Various models such as ISO 9126 or functionality, usability, reliability, performance and supportability (FURPS) exist to help here, but each organization should define its own standards for defining requirements and then use inspections and peer reviews to ensure that they are effective and used.

Although it may also be obvious that the architecture has also been defined, this is not necessarily the case. With development approaches that avoid the 'big design up front' approach, the architecture will be seen to emerge as development implementation proceeds. Clearly, the emerging architecture would eventually meet the requirements of architectural design.

The outcome of the Development Implementation process is a product that requires no unexpected rework. Therefore, organizations will aim to ensure that the product meets its requirements and, in addition, aim for there to be no unexpected defects or rework required. Failure to achieve either of these aims may give the organization additional effort and cost in the realization of the product.

The organization should have in place mechanisms for ensuring that the requirements are met and for dealing with defects and rework. In addition to ensuring that the right product is delivered, these mechanisms should also provide useful information about the performance of the management system, and should be used periodically to help identify improvement opportunities (see ORG.5 Improvement).

There are five base practices defined in the BPL Development Implementation process that work collectively to achieve the process outcome. These are:

BP.1 Establish the Development Environment
BP.2 Identify Component Sources
BP.3 Design Components
BP.4 Implement Components
BP.5 Manage Design and Implementation Changes.

While normally there is no implied sequence to the base practices in the BPL processes, some of the practices clearly work better after others have been completed. For example, without the development environment, no development is possible, and it is usual to design components before implementing them. Changes, however, can occur at any time, and are usually dealt with as they arise.

## *BP.1 Establish the Development Environment*

The development environment should be defined and established in line with the requirements of the project. This may be a standard environment, which is therefore constantly established and needs no special effort, or it may be a specific environment for a particular project, in which case specific care and attention needs to be given to its design.

In establishing the development environment, various aspects should be considered, such as design support tools, development tools, source code control and configuration management (document and code), group development environment, test tools and tools to support the release process.

If the environment is being set up especially for the project, then the activity should be managed as a key work package, including specifying the environment, planning and tracking its creation, and testing that the created environment is the one required.

Where the development environment is a standard one, its configuration still needs recording as part of the configuration management activities, although this may be done once only at the organizational level.

Where the project team is distributed, consideration should be given to the use of collaboration tools, such as conferencing tools, community repositories and project rooms.

Part of setting up the environment is ensuring that the people who are going to use it are trained and capable of using it effectively. Typically, a technical lead will just check that everyone has used the tools required on the project and arrange any training needed for those who have not or who need a refresher. They will also check that people have login information and the correct levels of access and privilege as required.

If any special resources are needed (e.g. a particular piece of test equipment, an expert in the particular technology), it is important that they are identified and secured, especially


 

where a resource is shared. Project delays due to the unavailability of shared resources are quite common, and very frustrating.

## *BP.2 Identify Component Sources*

There are three basic component sources resulting from buying, making or reusing. While a decision may have been made on the overall development approach at a high level, there can be further make, buy or reuse decisions at a component level.

Within the basic three choices there are further options. For example, possible reuse options are to:

- reuse a component that has been designed for reuse (low effort);
- take an existing component not designed for reuse and use it, but on this product only (more effort); and
- take an existing component not designed for reuse, make it generally reusable and then reuse it for this product (most effort).

Such choices would require suitable business justification. Similar gradations exist with regard to make and buy, and the project team needs to analyse these with regard to the effect they have on the system requirements, especially on the delivery time and budget.

When choosing the component source, consideration must be given to the full lifetime of the product or product family. Sourcing a component externally may look initially favourable, but when support for existing and changing versions is factored in, and perhaps extended to cover the lifetime of a family of products, the initial analysis may not be so clear.

Consideration should also be given to support costs beyond initial acquisition for the full lifetime of the product or product family. How will product components be supported in production, and in the field? This is as true for an internally sourced component (e.g. from another project team or product team) as it is for one sourced externally.

Externally sourced components or capabilities may give rise to the option to adjust the architecture to achieve time or cost savings. Such changes are handled through the change management process and lead, of course, to changes to the architecture and all derivative downstream products.

## *BP.3 Design Components*

Design transforms the requirements for the component to statements that can be implemented. It marks a key transition of the description of the solution from the language of the problem space to that of the implementation space.

The design of each component should describe how it operates, and include full details of its interfaces with the rest of the system and external world. It also includes details on which system requirements are being satisfied by the component, although such information may be controlled separately (e.g. through a traceability matrix). A combination of all the requirements detailed in the component descriptions and the interactions between the components should cover each one of the system requirements.

Where components are being sourced externally, the design should cover the interfaces between the component and the rest of the system, how they will be handled in the configuration management system, and how changes to the bought-in components will be transmitted through to the product. If architectural changes are considered necessary, the cost-effectiveness of the bought-in component may need to be reassessed.

The use of design methods, covering activities, diagramming techniques and support tools helps to improve the consistency and quality of designs produced, by ensuring that the same type of information is produced for all design components, and making it easier to cross-check information between different components.

If design standards have not already been defined, it is useful to consider them before starting design work. Consideration should be given to items such as performance, security, user interfaces, internationalization, navigation, help facilities, reuse and portability (e.g. support of multiple hardware platforms).

Similarly, establishment of a defined approach to design will promote greater consistency of design, thus reducing design errors and oversights, and making the subsequent system building much easier and more trouble-free. Consideration should be given to items such as:

- design method (e.g. procedural, object-oriented)
- concurrency (locking strategies)
- architectural layering
- type of database (e.g. object orientated, relational, flat file) and persistence strategy


 

- processing distribution (e.g. client–server, use of stored procedures)
- data availability, persistence and security
- resilience (error handling, communications and database recovery)
- messaging and interface communication
- data distribution (location, integrity, volumetrics).

Prior to commencement of any coding, the design should establish the following:

- external interfaces (e.g. graphical user interface (GUI), report layouts, file formats, remote calls)
- storage requirements (e.g. data model)
- links between the GUI, business process and database
- interfaces to existing infrastructure within the project.

These items may be captured informally (e.g. in existing code), or more formally in a design description, but in either case it should promote questions, discussions and improvement.

A formal technical review of the system design will help to minimize the defects passed through to subsequent phases. The review should check that every system requirement has been captured, that organizational standards with regard to architecture, communications and security have been met, and that there are no identifiable design defects (see TEC.4 Verification).

Organizations should review component designs and remove any defects found, and further analyse the causes of the defects in order to propose any necessary change in the process to prevent their future reoccurrence (see ORG.5 Improvement). However, the formality of the review process is best adjusted to the circumstances of the project or organizational area.

## *BP.4 Implement Components*

Components are implemented in the manner appropriate for the type of component (e.g. coding software, documenting services). In line with the approach to minimize defect leakage, the organization should include a check on the quality of the implemented unit (e.g. code inspections and unit testing for software, peer review for service descriptions).

The organization should also analyse the nature of defects found and work to remove their influence from future developments. The use of organizational implementation standards (e.g. coding standards) helps to prevent known errors from occurring during implementation, as does the use of coding support tools.

Where there are multiple development teams implementing components, the control of different versions of components and their integration into a stable configuration assumes greater importance than for co-located teams. This is why organizations strive to create 'four-wall' teams but are often influenced by desires to use more cost-effective 'off-shore' resources.

Acquiring components externally is not just a case of receiving them. The following items should be considered:

- the activities that should exist to ensure that the acquired component is defect-free
- the design information that should be acquired
- the maintenance and support information should also be acquired
- how the components will be managed under control conditions
- how changes to the components will be handled
- how problems with and defects in the acquired components will be handled.

## *BP.5 Manage Design and Implementation Changes*

Changes are a fact of life, and in some lifecycles are actively encouraged. Changes can come from outside the organization (e.g. a change in legal requirements), outside the project or organizational area (e.g. a change in business requirements) or from within the project or organizational area (e.g. from discovering a defect).

Once top-level design is approved and development work has started, changes to any of the key configuration items (e.g. customer requirements, system requirements, top-level design) may occur. Such changes must be carefully managed. The mechanism for this may vary from a discussion and agreement in, for example, an agile team, to a formal change management system (see PRJ.3 Configuration and Change Management).


 