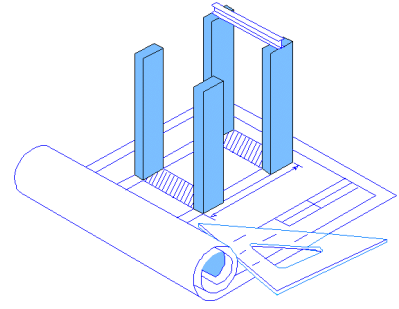


TASSC

technical paper



ObjectMetrix Process

Gillian Adens, Robert Armstrong

Tassc Limited
www.tassc-solutions.com

First Published: December 2001
Last Updated: January 2008

Copyright 2001-2010, Tassc Limited. All Rights Reserved.

Every software project follows a development process, whether it is informal and unstructured, or formal and documented in detail. A software development process is essentially an approach to organising and managing the various stages in a project to successfully deliver a software system within a predictable timescale and to an acceptable budget, in a measurable and reliable way.

A development process is fundamental to software engineering. Projects that are to be managed and controlled effectively need appropriate mechanisms to ensure a consistency of approach and to ensure the quality of the deliverables. The purpose of the ObjectMetrix estimation process is to provide guidance on what estimates need to be produced, when they should be produced, who should be involved and how software development organisations can benefit from the results.

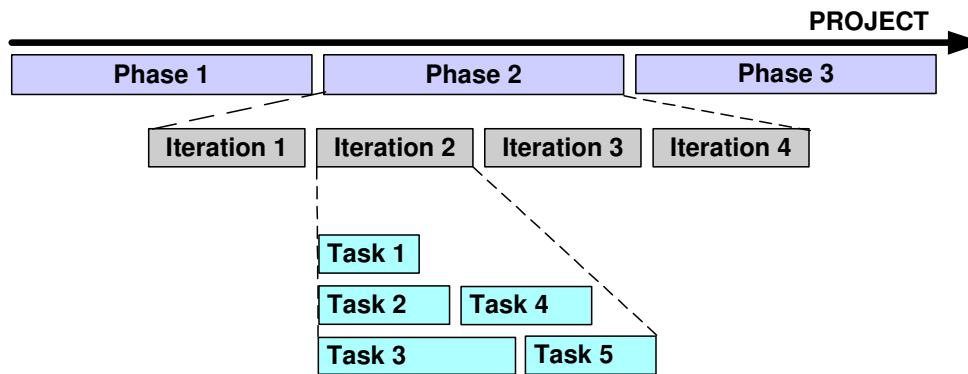
Modelling the software development process

*companies mix
and match, and
customise
development
processes*

A number of well-defined software development processes exist and are published widely e.g. Dynamic Systems Development Method (DSDM), Rapid Application Development (RAD), the Rational Unified Process (RUP) and the traditional Waterfall process. Organisations tend to customise these 'off-the-shelf' offerings to better suit their own environment, standards and way of working. Most large organisations will follow more than one development process as each is better suited to specific types of development e.g. RAD might be very appropriate for prototyping new user interfaces whereas a more structured process like RUP might be preferable for the development of a company-wide generic framework. Strenuous

*ObjectMetrix
defines a
generic process
model*

Despite the variety and apparent diversity of software development processes most share similar fundamental concepts. Each development process differs in the order and level to which the various activities are tackled and each may define and use its own terminology. In order to support the most widely adopted concepts and provide a practical and reliable framework, ObjectMetrix defines a generic process model which can be customised to capture industry best practice.



a process comprises phases, iterations and tasks

In ObjectMetrix, each project follows a process. A process comprises a number of phases. Phases comprise a sequential set of iterations within which tasks can be defined. Tasks can be scheduled and completed in parallel or sequence. Within each stage of the project a number of software development and related activities are undertaken either to completion or to a well-specified intermediate level.

obtain reliable estimates at a more granular level

With a process in place, ObjectMetrix allows managers to obtain reliable estimates at a more granular level, predictably and repeatedly. ObjectMetrix provides detailed estimates for a specific phase, iteration or task. Given project status information, ObjectMetrix can be used to calculate effort, time and cost to completion of a particular iteration, phase or for the project as a whole.

Process

a process provides a framework for development activities

A development process provides the necessary guidelines and framework for development activities, defining the stages of a project, appropriate activities during each stage and the results expected at the end of each stage. A process provides the necessary checks and controls to ensure that diverse development roles and activities are orchestrated to deliver a quantifiable result in terms of artifacts that contribute towards the eventual software solution.

enables a standard approach

A process defines a particular approach to software development that has proven suitable and successful for specific types of software construction. A set of criteria can be defined to indicate whether a particular process is appropriate for a given project. A process defines the rules and guidelines that enable a standard approach within an organisation.

encourages constructive feedback and process improvement

The process defines the nature of the activities to be undertaken, a suitable order for tackling the various activities, the methods or techniques appropriate for each activity and the expected timescales and deliverables from each activity. Ideally, a process should also include constructive feedback and process improvement mechanisms.

Phases

phases are the major stages of a project

Phases are the major stages that a project progresses through. Examples of phases might be feasibility, requirements analysis, architecture, production increments, deployment and maintenance. Each phase occurs at a particular point in the overall project lifecycle and each has a different primary focus. In general, early phases during the formative stages of a project tend to be more creative and involve activities such as prototyping and requirements gathering. Later phases move into software production and involve activities such as coding, integration and testing. Phases are sequential and follow a logical order.

*early phases
explore
concepts and
assess feasibility*

Typically software comes into existence from an idea or need. Some initial work is carried out to explore the concepts and assess the viability of proceeding by verifying technical feasibility, cost justification and timescales. An overall project estimate based on project scope addresses this important decision point.

*high level of
uncertainty and
risk*

During the early phases of a project such as inception or feasibility a key estimation factor is a distinct lack of concrete information and a high level of uncertainty and risk. At this stage strategic management is principally interested in the business implications of the proposed system. In terms of estimation this is always reduced to overall budget and timescales.

*gather
requirements
and define
architecture*

Having made the decision to proceed with the project, emphasis moves into detailed requirements definition often involving end users, customers and business analysts. As requirements stabilise and become clearer, technical considerations such as the overall structure or architecture of the software system need to be defined.

*completion of
each phase
represents a
major milestone*

The completion of each phase of a project represents a major milestone in the overall project lifecycle. It is important to estimate the effort, duration and cost of individual phases and equally important to continually monitor time to project completion taking into account the progress already made through the previous phases of the project.

Iterations

*projects iterate
over a set of
activities to
deliver a solution*

Phases are comprised of iterations. Software projects iterate over a set of activities to deliver a solution e.g. a working piece of software, often called an increment. Each iteration has a clear set of goals and objectives and results in a set of deliverables. The deliverables might be the completion of fully operational software or the manifestation of an artifact such as a new version of software documentation.

*the number of
iterations
depends on the
scale of project*

Where a process generally has a fixed number of named phases, the number of iterations is variable and depends on the scale and nature of the project. The completion of a phase can be measured by progress through the various iterations within a phase. Completion of each iteration is a significant milestone within a phase.

*parallel streams
converge to a
single cohesive
solution*

Although iterations themselves are sequential, individuals and teams can work in parallel within an iteration. There are often parallel streams of activity going on during software development such as production of on-line help and user guides. However, eventually these activities converge to the release of a completed increment. Each iteration starts from a known baseline, builds from this by carrying out a variety of development activities resulting in deliverables which are integrated, reconciled and synchronised to converge to a single cohesive solution.

Tasks

*a task is a unit of
work that can be
assigned to
resources*

Within each iteration a number of tasks can be specified. A task is a unit of work or work package that can be assigned to one or more individuals. Tasks are the application of resources to evolve software applications i.e. tasks indicate which aspects of the software individuals are assigned to work on.

*software
development
effort is divided
by resource
availability*

During the construction phase of a project, tasks, for the most part, relate directly to the development of software artifacts. Therefore a task can be defined by those software artifacts to be constructed. The effort associated with the software construction process for those artifacts can then be divided amongst the assigned resources according to their availability.

tasks can be scheduled and completed in parallel or sequence

Tasks can be scheduled and completed in parallel or in sequence. There are often clear dependencies between tasks, which need to be taken into account by the project manager during the scheduling activity. Each task will typically require iteration over a number of development activities. Completion of tasks represents an easily quantifiable progress measure within an iteration.

When to estimate

different estimation purposes

Within any software project people perform different roles and are therefore interested in estimating for different purposes at various points in the software development process.

feasibility at early stages

To establish project feasibility senior project managers need to calculate overall project estimates of effort, duration and cost based on known requirements statements. At this stage the management team needs to derive information from the conceptual definition of the project and apply appropriate algorithms that take account of their level of abstraction. The result is a first-cut estimate for the overall project, which provides a global view to senior project managers. This is often called the 30,000ft view of the project.

high-level project plans

As the project progresses through the development process, business analysts, technical architects and team leaders can develop a more detailed project definition from the results of requirements analysis and initial work on the software architecture. At this stage the scope specification is more accurate but still at a level of abstraction above the implementation details.

detailed schedules

Finally as the project moves into production iterations, and detailed design is fleshed out, software engineers and team leaders require highly detailed estimates for tasks based on a more concrete definition of actual software artifacts to be constructed.

estimation is a process of refinement

Like many problem-solving techniques, estimation is a process of refinement. The more you discover, the clearer the solution becomes, and with ObjectMetrix, estimates become increasingly more accurate and reliable.

Level of granularity

granularity increases over time

In ObjectMetrix the level of granularity of a software specification moves from conceptual (the high-level analysis model) towards a concrete definition of project scope (low-level design model).

identify concepts then discover concrete software artifacts

At the exploratory stages of a project the specification will remain sketchy. The basic concepts can be identified but an analysis and design process will be necessary to refine these conceptual software artifacts. Gradually these concepts will be further decomposed resulting in the identification of one or more concrete software artifacts. Concrete software artifacts directly correspond to items to be developed and coded.

requires different concept and concrete metrics

It is relatively straightforward to obtain accurate estimates for concrete software artifacts but the further removed from this level of detail of the system scope, the more challenging the problem. It is still possible to get a first-cut estimate at the conceptual stage provided appropriate metrics are used and reasonable contingency is applied.

concept metrics provide early predictive estimates

ObjectMetrix provides a base set of productivity metric values for both concept and concrete software artifacts. Concept metrics are used at the stage in the project when initial requirements analysis is complete but software construction has yet to commence. These metrics are valuable as they provide an up-front predictive estimate allowing earlier creation of accurate project plans and schedules.

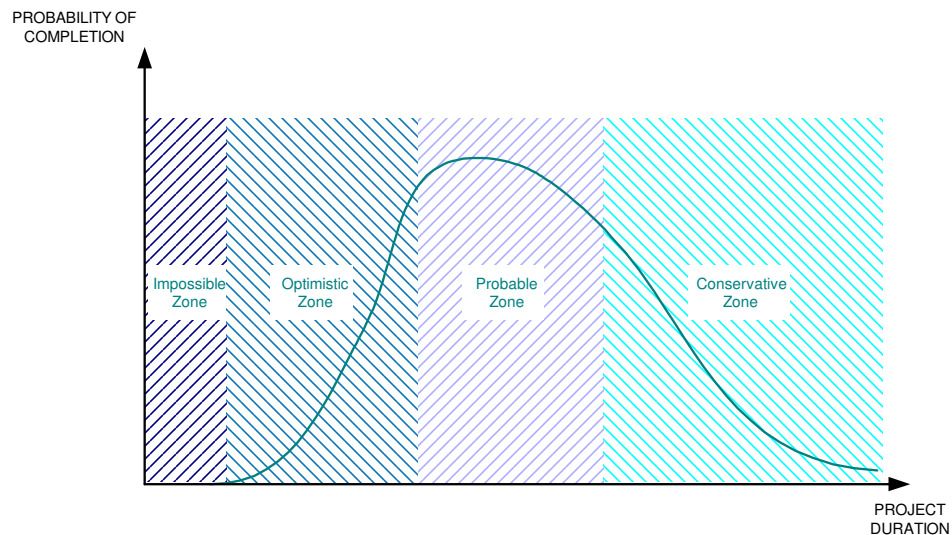
concrete metrics
provide more
accurate
estimates

Concrete metrics can be used to obtain an estimate at the detailed design stage of the project. By this means overall project estimates and detailed iteration or task estimates can coexist and can both be accurate, equally valuable and useful. Concept software artifacts can be realized as concrete software artifacts at any time during the software development lifecycle, providing an increasingly accurate estimate. Typically one concept artifact will become one or more concrete artifacts.

Bounds on estimate accuracy

build in an
appropriate level
of contingency

When estimating from a conceptual project definition the margin for error is at its greatest and reduces as the scoping exercise moves towards a concrete definition. For estimation purposes the margin of error will reduce over time. By building an appropriate level of contingency into the estimates, depending on the granularity of scope specification, results can be presented with a higher degree of confidence. At the early stages of a project it is often more meaningful and helpful to present results as an estimation range rather than a single point estimate.



a project is more
likely to lose a
week than save
a week

Theoretically, a graph can be plotted to map the probability of a project completing against project duration. To the left is the 'impossible' zone. Given the amount of scope, the project simply cannot be completed this quickly. The start of the curve is the earliest possible delivery date, assuming the project runs absolutely smoothly. This is unlikely, but possible. Moving to the right the graph curves upwards - these later dates are increasingly more probable. At its highest point the graph indicates the most probable delivery date, and thereafter the line slopes gradually down again. The curve is considerably steeper on the left than on the right. This is because starting from the most probable date, taking a longer period of time is more likely than taking an equivalent shorter time. A project is more likely to lose a week than to save a week.

choose a
realistic delivery
date

There is evidence that the vast majority of projects aim for an 'optimistic', below-average target date. But by definition, below average is the exception rather than the rule. The more optimistic the target, the less likely the project will succeed in hitting it. This easily leads to a false perception of failure. It would be more sensible to choose a date in the 'probable' zone that splits the area under the graph evenly. However, if the project cannot afford to deliver late then it would be wiser to err on the side of caution and predict a date in the 'conservative' zone.

Who should estimate

*planning,
scheduling,
managing or
controlling
software
development
projects*

ObjectMetrix supports anyone involved in planning, scheduling, managing or controlling software development projects – in fact just about every role in software development:

Customer

Prioritise product functionality and negotiate delivery dates

Bid Manager

Calculate development costs and timescales and produce a competitive bid

Business Analyst

Analyse requirements and scope the development effort

Technical Architect

Compare and contrast alternate technical solutions, estimate effort to build common infrastructure

Programme Manager or Senior Project Managers

Establish project feasibility, authorise project resources and budget

Project Manager

Produce overall project plans, define project milestones and delivery dates, track overall project progress

Team Leader

Produce detailed schedules, define scope of each software iteration, track task and goal completion

Software Engineer

Estimate effort for individual work packages

Quality Assurance Manager

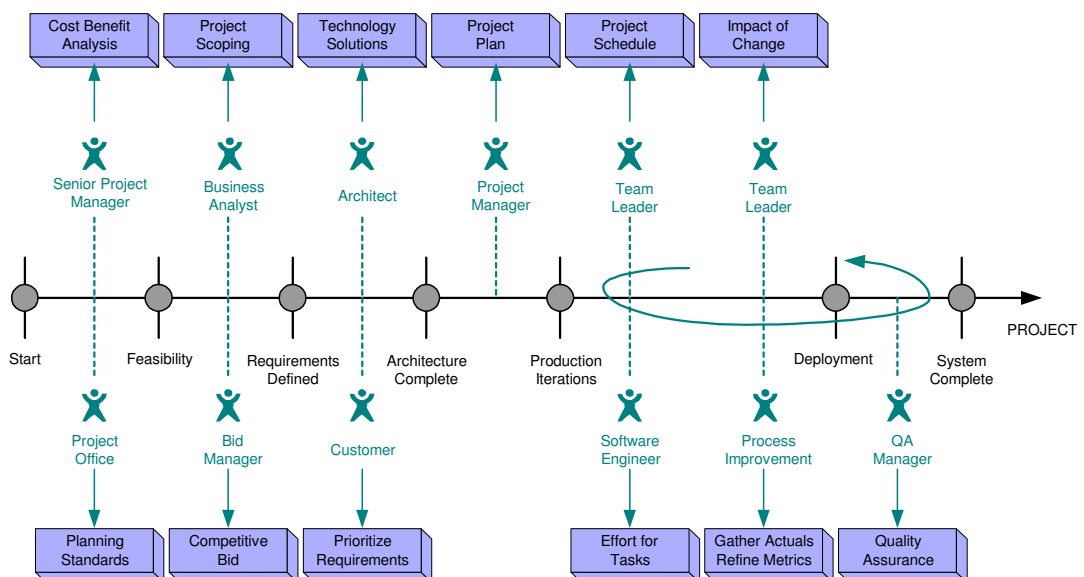
Ensure time allocated for testing, drive test plans from project scope definition

Process Improvement Team

Ensure a consistent estimation technique, implement a company-wide metrics-gathering programme

Project Office

Define a consistent approach to project definition, planning and scheduling



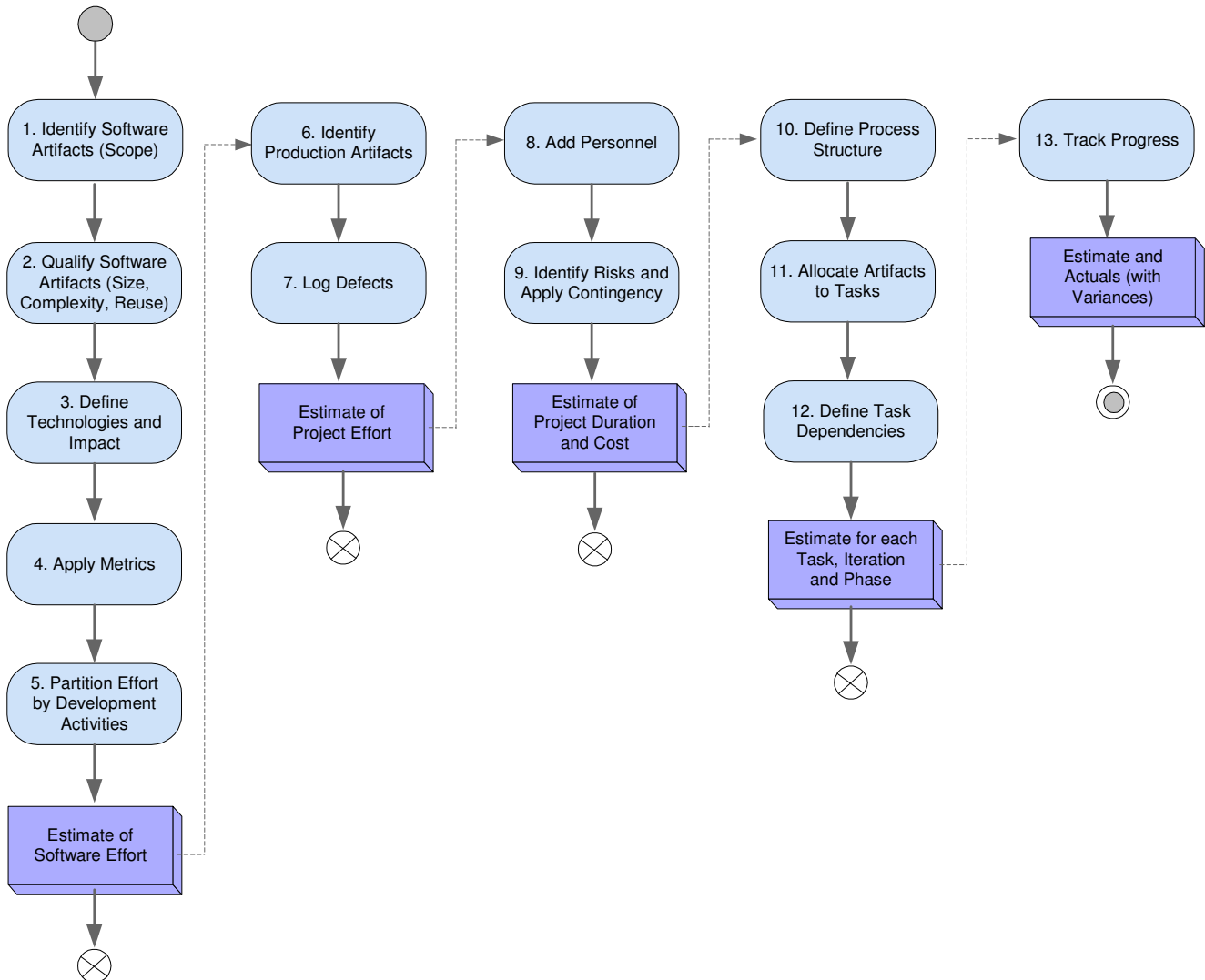
The ObjectMetrix model

many diverse factors affect project duration

Many diverse factors have the potential to affect the duration of a development project. Among them are insufficient budget, scarcity of resources, the sheer amount of work to be done, pre-defined deadlines, technical risk, team skill and the inherent difficulty of the problem domain. Tasked with managing the complexity and dynamics of software development projects, the interaction of these and other factors must be adequately addressed. Such factors are often inter-dependent, and consideration of any single factor cannot be done in isolation.

ObjectMetrix estimation model

The ObjectMetrix estimation model defines a process for estimating that incorporates a set of significant project factors necessary to measure and assess the project as a whole.



1. Identify Software Artifacts (Scope)

clearly identify those software artifacts to be constructed

The first step in estimating is to establish the scope of the project – to clearly identify those software artifacts to be constructed and establish the boundaries of what is considered within and outside the scope of the project. Project scope is essentially a measure of scale and for the purposes of ObjectMetrix estimation can be defined in terms of the Unified Modeling Language (UML), a standard notation for designing object-oriented, component-based and web-based software systems.

an approximation will render an estimate that is useful for high-level project planning

At the outset of a project, when the requirements are being explored, it can be a challenge to accurately identify everything that may be eventually constructed in software. However, even a close approximation of project scope will render an estimate that is extremely useful for high-level project planning. As more requirements analysis takes place, and detail is fleshed out, an increasingly accurate result can be obtained.

agree the system functionality as a set of use cases

Generally the first stage in scoping a project is to agree the system functionality – often as a set of use cases. Unfortunately there is no proven correlation between the number of use cases and the number of classes or subsystems. In a given application domain it may be possible to generalise and use ratios to obtain a first-cut estimate. However, in the general case it is not possible to provide metrics for these ratios as systems vary enormously from commercial or query based systems that are highly functionality biased to back-end engines which provide a limited interface but implement complex calculations using a large number of specialist classes.

it is surprising just how much business information can be obtained

It is therefore dangerous to generalise, and evidence shows that it is very worthwhile to spend just a little more effort at the requirements analysis stage to identify classes and subsystems as well as use cases. Although this may sound like a lot of additional work, it is surprising just how much business information can be obtained from a one-line description of each use case. Such an exercise will highlight the vast majority of business concepts that will be realized as software classes. Similarly a rough grouping exercise by end user role (actor) forms an excellent first approximation for subsystems.

2. Qualify Software Artifacts (Size, Complexity, Reuse)

qualifiers provide a qualitative judgement to refine the estimate

Qualifiers provide a qualitative judgement on each software artifact that allows a refinement to the estimate. Qualifiers are factors that refine project scope in order to account for aspects that are unique to a particular project, and depend upon the development strategy and the problem domain. Qualifiers are qualitative and therefore more subjective than the quantity of project scope.

ObjectMetrix supports four qualifiers:

Size – the sheer amount or scale of work to be done

Complexity – the algorithmic diversity

Reuse – the ability to benefit from pre-existing software

Genericity – the intention to build general purpose software for reuse

a qualifier mask indicates the impact on each activity

Qualifiers refine the effort in each activity by adding or subtracting a delta, by way of a percentage, to each activity estimate. The qualifier algorithms in ObjectMetrix model the impact of multiple qualifier values. Rather than simply aggregating the effects of qualifier values, the effects are multiplied to produce an interrelated cumulative result.

If the qualifier levels that are selected are each in turn applied to increase the amount of effort, the effect will be cumulatively greater to simulate the impact of compound risk e.g. to reflect the likelihood of very large, highly complex, and extremely generic solutions taking substantially more effort to construct. If the qualifier levels that are selected are each in turn applied to decrease the amount of effort, the effect is to minimise the effort without it ever diminishing to zero e.g. even the smallest, most trivial software artifact with high reuse will still require some minimal effort to construct.

3. Define Technologies and Impact

programming languages, tools and techniques or development environments

In ObjectMetrix, the impact of technology can be defined for specific programming languages, tools and techniques or development environments.

The choice of technology affects the amount of effort required to develop software. Contrary to popular belief, the overall effect on the estimate is often small, generally much less than that of qualifiers or the skill level of the development team.

technology has a percentage impact on certain activities

Like qualifiers, technology has a delta impact on development activities. A technology mask indicates those aspects of the activity profile that are affected and by how much, by stating an impact percentage for each activity. The qualified estimate is technology independent and represents an average effort. Applying a technology mask accounts for the specific characteristics of a chosen technology.

4. Apply Metrics

metrics provide a standard of measurement

Metrics are used within the ObjectMetrix model as a standard of measurement for the effort required to develop software artifacts. Clearly software artifacts are not developed in isolation. In fact a use case cannot be realized without building classes. Equally a subsystem is the packaging of a set of use cases to implement a cohesive user interface or application. However, by applying a base productivity metric to each software artifact according to its classifier type it is possible to predict the effort involved in developing a complete software system.

the number and type of each software artifact provides quantitative data

The number and type of each software artifact provides quantitative data that is key to determining the resulting project estimates. A first-cut estimate can be obtained by summing the number of software artifacts of each type and multiplying by the related productivity metric. This value is then refined by applying qualifiers and technologies to each software artifact in turn.

5. Partition Effort by Development Activities

a full life-cycle estimate

ObjectMetrix provides a full life-cycle estimate, which accounts for the wide range of development activities required to build well-engineered, well-documented, maintainable, quality software. Time allocated for each software artifact is apportioned across this range of development activities.

planning, analysis, design, build, testing, integration and review

The software development process is non-trivial and as software becomes increasingly more complex and sophisticated significant emphasis is placed on a proper development process. Development lifecycles vary but software construction is characterised by a number of key software development activities, namely: planning, analysis, design, build, testing, integration and review. The total effort to develop each software artifact is apportioned across these activities.

an activity profile divides the productivity metric into effort for each activity

An activity profile divides the base productivity metric into effort for each activity. The default activity profiles provided by ObjectMetrix assume an iterative development lifecycle. These defaults can easily be tailored to suit the style of development adopted within a particular organisation.

The activity profile can be applied to the base productivity metric to result in a set of activity estimates for each software artifact. The combination of productivity metrics and activity profiles allows the accurate breakdown of overall effort into effort on each development activity. This provides a good indication of the mix of skills required in the development team.

6. Identify Production Artifacts

a product is not software alone

A software development project is a complex process that requires careful planning and scheduling. In addition to software construction there are generally other related activities which are preliminary, supporting or by-products of the software development effort. There may be end-user documents to produce, hardware to build and integrate, help systems to construct, text and graphics to compose, product icons and branding to be designed and any number of other specialist non-software activities to be scheduled.

incorporate production artifacts to get a whole project estimate

Production artifacts are specialist in nature, require specific sets of skills that may lie outside the software engineering team, and require a significant degree of effort. In order to obtain an accurate estimate for the project as a whole, these production artifacts need to be accounted for alongside the software construction activities.

7. Log Defects

record defects to allow the development team to reproduce and rectify any problems

It is almost inevitable that defects will be found both during development and after release. Some problems are reported by engineers assigned to test the product prior to release, others are found by customers whilst using the product. Wherever and whenever these defects are found they need to be recorded in enough detail to allow the development team to reproduce and rectify any problems.

access defects in terms of severity and prioritise their investigation

Once problems are identified and recorded, it is necessary to access them in terms of severity and prioritise their investigation. Defect severity can be recorded as trivial, minor, important, major or critical. An estimate is made of the effort required to investigate and rectify each software and production defect. Defects can then be scheduled for investigation and resolution. A defect can be in one of six states – logged, accepted, rejected, in progress, resolved and verified.

8. Add Personnel

team size impacts duration

The team size is a significant factor in establishing project duration. Software development remains a labour intensive activity. Where there is a certain amount of effort required, clearly two or more developers will complete the task more quickly than one. Team members can co-operate and delegate activities enabling the team to perform better and more efficiently than any individual.

as team size increases, communication and reporting overheads increase

This is true to a point, however as the team size is increased, the communication levels also increase and channels of reporting become more formal. A small team can communicate on a fairly informal basis whereas a large team will require more formal documentation and regular formal meetings. Therefore adding additional developers will decrease duration until the inter-personal communication and management overhead outweighs the advantage of further partitioning the work. ObjectMetrix can assist in the calculation of the optimal team size.

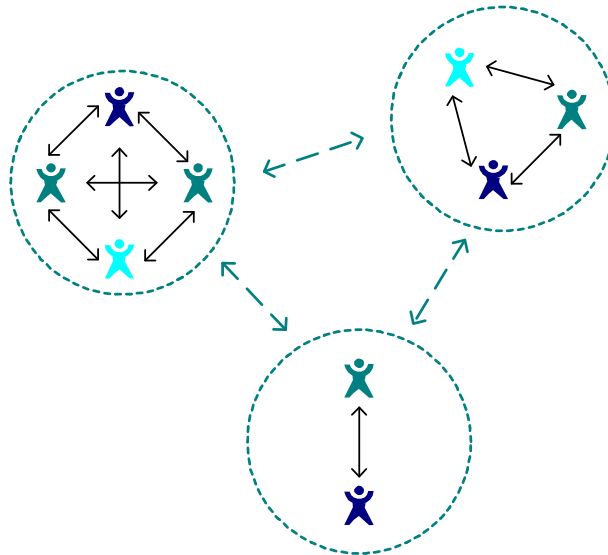
separate development teams can reduce the lines of communication

As a general rule the larger a development project, the more realistic it is to segment development activity and construct various aspects of the software in parallel. Where it is feasible to implement the software in a modular fashion a number of separate development teams can focus on different sets of requirements and be tasked to construct software in relative isolation. Again this can have a positive effect on duration by reducing the lines of communication between individuals to that of team-to-team interfaces.

the noise factor accounts for the level and quality of communications

Inter-communication and management overhead is calculated for each team assignment. Three aspects of the communication are taken into account – duration, frequency and quality. Duration is how long the communication lasts. Frequency is how often communication occurs both formally and informally. Quality is the level of understanding and information that is exchanged.

ObjectMetrix takes this into account by a concept termed the noise factor that maps the level and quality of inter-personnel communication based on the size and skill level of project teams, and thus provides a non-linear equation to calculate duration from effort.



9. Identify Risks and Apply Contingency

reduce or eliminate the negative impact of problems

Risk management is about reducing or eliminating the negative impact of potential problems. Rather than passively reacting to events, those responsible for managing risk need to be proactive, vigorous and diligent, both in advance and during the software development process.

describe risks in terms of their impact and probability

Risks are described in terms of both their impact and their probability. Each risk can be assigned a probability of firing and a level of impact. The level of impact is the amount of damage the risk will have on the duration and cost of the project, should it fire. The impact of a risk can be recorded as minor, low, high, severe or critical. The probability of firing indicates a judgement on the likelihood of the risk firing given the project circumstances. Options include remote, unlikely, 50/50, possible or probable.

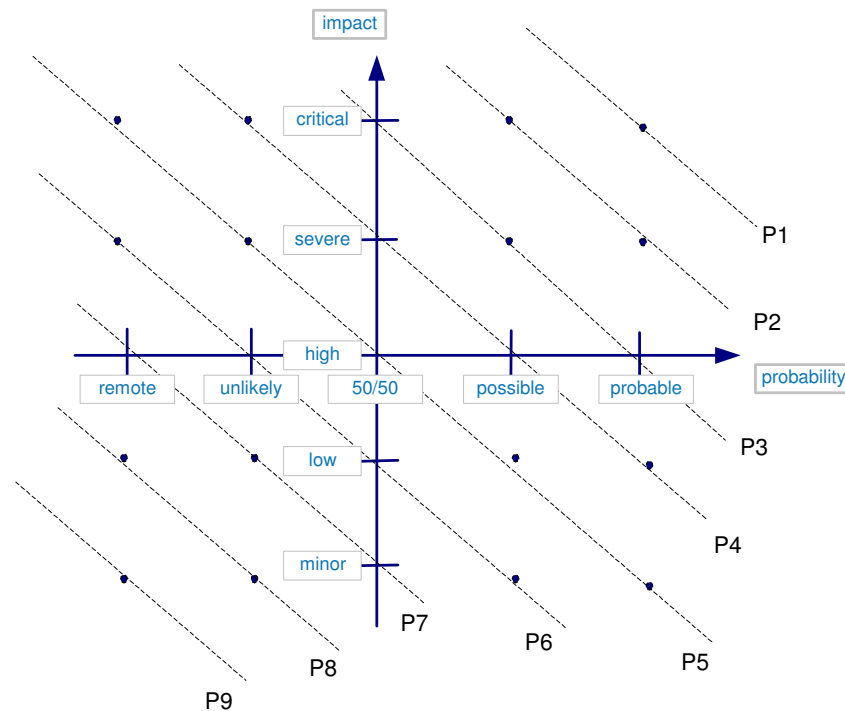
calculate a contingency for the project

The combination of probability and impact defines the level of threat from each risk to the project. This is the crucial factor in calculating an appropriate contingency for the project. Contingency allows us to estimate a range of durations and plot realistic bounds on the estimate calculations. The earliest possible duration is where no risks fire. The latest possible duration, where all risks fire, is calculated by summing the potential impact of each risk. A third contingency value can be derived from averaging the risk threats – this provides a sensible level of contingency based on the combination of all risk impacts and probabilities.

*develop
contingency
plans for priority
risks*

Risk priority ranks the risks by their level of threat to the project. It is driven by the combination of risk impact and probability. It is essentially an indicator of the level of effort that should be put into resolving each risk. The high priority risks are those that we should actively try to resolve, and that we need to develop contingency plans for, so that we have an alternative action or solution should they fire.

A simple way of defining the risk priorities is to plot the risk impacts against probabilities, and draw priority lines diagonally through the graph. Those risks of highest probability and impact are of greater priority and those risks of least probability and impact are a lower priority.



10. Define Process Structure

*define the
software
process in terms
of phases,
iterations and
tasks*

In order to schedule work the project manager needs to define the software process as discussed earlier in this paper – as a chart containing phases, iterations and tasks with appropriate dependencies between each of these. Each phase, iteration and task will have a clear and well-defined goal within the context of the overall project. The nature of the process will depend on the organisational goals and technologies in use. However, most UML projects elect to use an iterative process – allowing the software to evolve over time.

11. Allocate Artifacts to Tasks

*allocate related
artifacts and
defects to tasks*

ObjectMetrix allows software artifacts, production artifacts and defects to be allocated to tasks in order that they can be assigned to workers for development. Obtaining a sensible allocation of artifacts can be a time-consuming activity, as ideally it should reflect the dependencies and relationships between them. The ObjectMetrix approach is to drive the process from a functional point of view. This might involve building a matrix map of dependencies between software artifacts, which allows the project manager to more easily carve the scope into related groups for development. For example, a number of related use cases and the classes that implement those use cases could be assigned as a single cohesive and meaningful task. The architectural structure of the software artifacts provides an excellent start point for this activity.

software and production artifacts are typically partitioned across a number of tasks

It is rare for a software artifact to be fully analysed, designed and built in one go. It is more common for each software artifact to evolve in stages. Perhaps the initial analysis work is undertaken by business analysts and senior designers, then the software engineers adjust the specification to meet the technical design constraints of the software architecture and non-functional requirements, then the software is built and tested in stages (the core or high priority functionality first). Likewise, production artifacts can also be completed incrementally. Therefore artifacts are typically partitioned across a number of tasks, and possibly across iterations.

12. Define Task Dependencies

task dependencies will impact project estimates

There are often dependencies between tasks that need to be modelled in order to obtain an accurate project estimate. Some task may be able to progress in parallel but others may not be able to commence until certain activities have been completed – thus elongating the schedule. With work allocated to tasks and task dependencies defined, ObjectMetrix can provide estimates at a more detailed level. The effort, duration and cost can be calculated for each task, iteration and phase.

13. Track Progress

estimates, actuals and variances for each task, iteration and phase

As the tasks are completed the actual effort, duration and cost can be documented. This historical information provides data for the calculation of variances, which feeds the refinement of metrics for future projects. Thus ObjectMetrix facilitates process improvement, and plans and schedules of increasing accuracy.

Conclusion

a reliable estimation technique is an essential ingredient in ensuring that schedules and budgets are realistic

In 1984, the United States Department of Defence formed the Software Engineering Institute (SEI) to establish standards of excellence for software engineering and to accelerate the transition of advanced technology and methods into practice. After years of experience with the software process and based on extensive feedback from both industry and government, SEI developed the Capability Maturity Model for software (CMM).

The CMM presents sets of recommended practices in a number of key process areas that have been shown to enhance software process capability. CMM defines 5 levels of software process maturity: Initial, Repeatable, Defined, Managed and Optimizing.

In order to progress through the CMM levels, organisations are required to establish methods and practices that are measurable and allow earlier project successes to be repeated in a controlled and predictable way.

A reliable estimation technique is an essential ingredient in ensuring that schedules and budgets are realistic. An estimation technique that allows actual metrics to be fed back offers the prospect of real process improvement. Using ObjectMetrix with metric data obtained from measurement within an organisation offers a high level of confidence that the expected results for cost, schedule, functionality, and quality of the software product can be achieved.