



Circa technical briefing

Interpreting and Calibrating Metrics

Gillian Adens

Tassc Limited
www.tassc-solutions.com

First Published: November 2002
Last Updated: February 2009

Copyright 2002-2010, Tassc Limited. All Rights Reserved.

Circa arrives ready loaded with metric data to assist the estimation process. Circa provides a default set of productivity metrics and activity profiles. This technical paper explains how these metrics are used in the calculations, and how you should interpret the results.

Furthermore Circa allows the metrics to be customised to refine and improve the estimation calculations. To do this effectively requires a detailed understanding of the underlying ObjectMetrix estimation algorithms – the purpose of this paper.

The ObjectMetrix model

*combination of
practical
experience and
research*

The ObjectMetrix estimation model is based on over a decade of practical experience in consulting on software development projects. This is combined with a theoretical foundation resulting from with an extensive 5-year research project.

*models
significant project
factors*

The ObjectMetrix model identifies a set of significant project factors:

Step 1 - Scope the application. Identify the software artifacts (use-cases, classes etc) to meet the project requirements.

Step 2 - Qualify software artifacts. Adjust size, complexity and reuse levels for each software artifact.

Step 3 - Define technologies. Indicate technologies used and their impact.

Step 4 - Apply metrics. Use built-in productivity metrics or define your own.

Step 5 - Partition effort by development activities. Use built-in activity profiles or define your own.

Result 1 - Produce estimate for software effort.

Step 6 - Include production activities. Identify production artifacts (help, branding, infrastructure, hardware integration, overheads etc.)

Step 7 - Log defects. Identify software and production defects to be addressed as part of the project.

Result 2 - Produce estimate of project effort.

Step 8 - Add personnel. Profile resources, create teams and assign resources to work within teams.

Step 9 - Factor in contingency. Identify risks, their likelihood and impact or manually override contingency.

Result 3 - Produce estimate of project duration and cost.

Step 10 - Define process structure. Create phases, iterations and tasks.

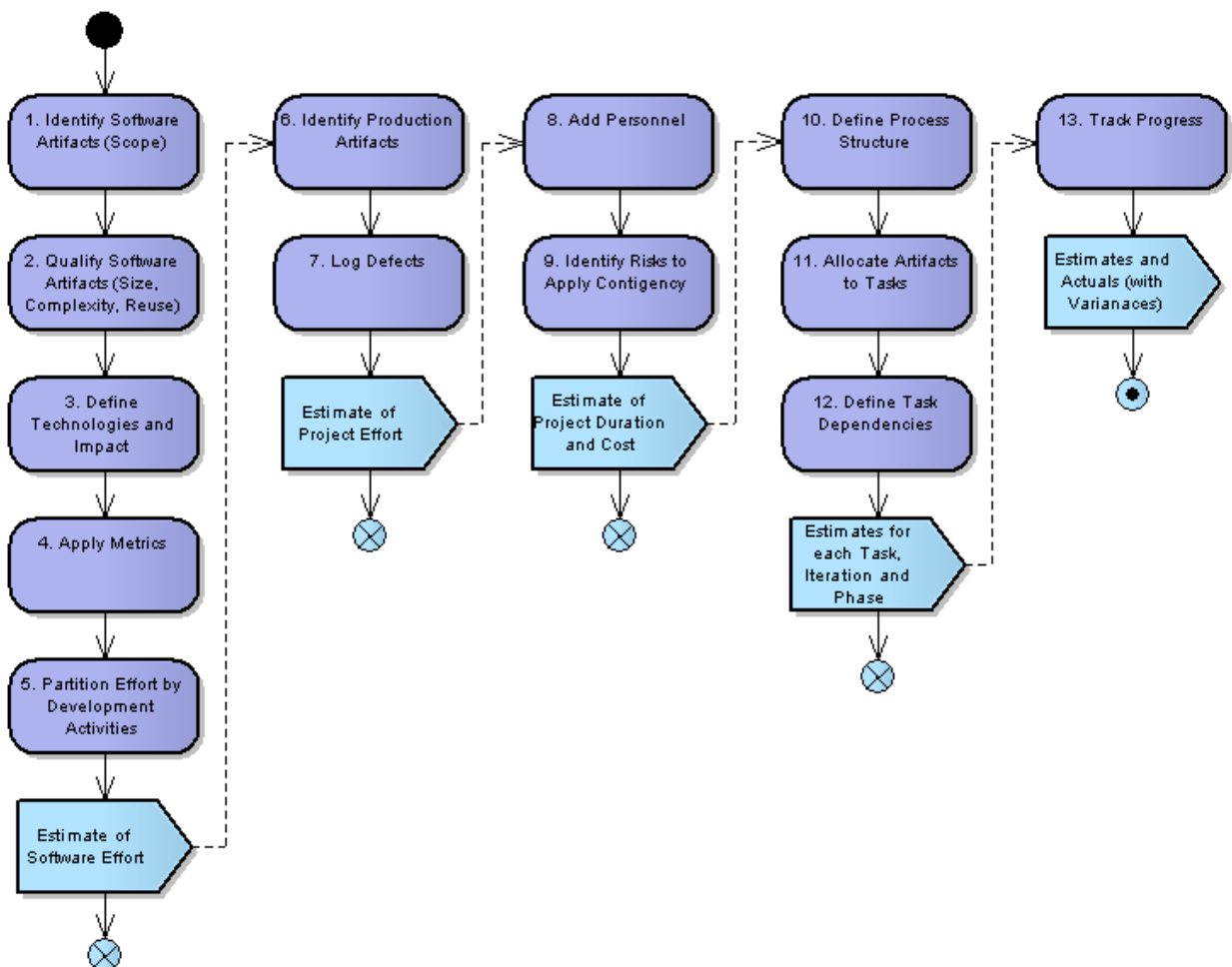
Step 11 - Allocate artifacts to tasks. Populate tasks with software artifacts, production artifacts and defects. Assign workers to tasks.

Step 12 - Define task dependencies. Indicate the order in which tasks should be worked on.

Result 4 - Produce estimates for each phase, iteration and task. View Gantt chart and resource utilisation.

Step 13 - Track progress. Set task status and record actuals.

Result 5 – Compare estimates and actuals. Analyse variances and adjust metrics for future projects.



What do the productivity metrics mean?

define and count standard UML constructs

ObjectMetrix defines and counts a standard set of UML constructs, namely: use cases, classes, subsystems, components, interfaces, web pages and scripts. These are the software classifiers (types of software artifact) that we use to measure the effort required for software development.

a standard of measurement for the effort to develop software artifacts

Productivity metrics are used within the ObjectMetrix model as a standard of measurement for the effort required to develop each of these 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, we can predict the effort involved in developing a complete software system.

built-in productivity metrics

By considering the typical development activities applied to each type of software artifact – during planning, analysis, design, build, testing, integration and review – we can define an appropriate productivity metric for each software classifier. In Circa the built-in productivity metrics are as follows:

hours effort	use cases	classes	subsystems	components	interfaces	web pages	scripts
concept metric	90	60	180	210	105	5	30
discovery metric	30	12	60	56	27	1	4
population ratio	1:4	1:6	1:1	1:1	1:3	1:2	1:2
concrete metric	15	8	120	154	26	2	13

analysis and design metrics

The metric applied will depend on the stage in the development lifecycle. If the scope is expressed as an abstract high-level analysis model – then concept metrics are used to reflect the likely scope population increase during analysis and design.

If scope is expressed as a detailed design model, then concrete metrics are used to capture the design and build activities required to turn this design into working software.

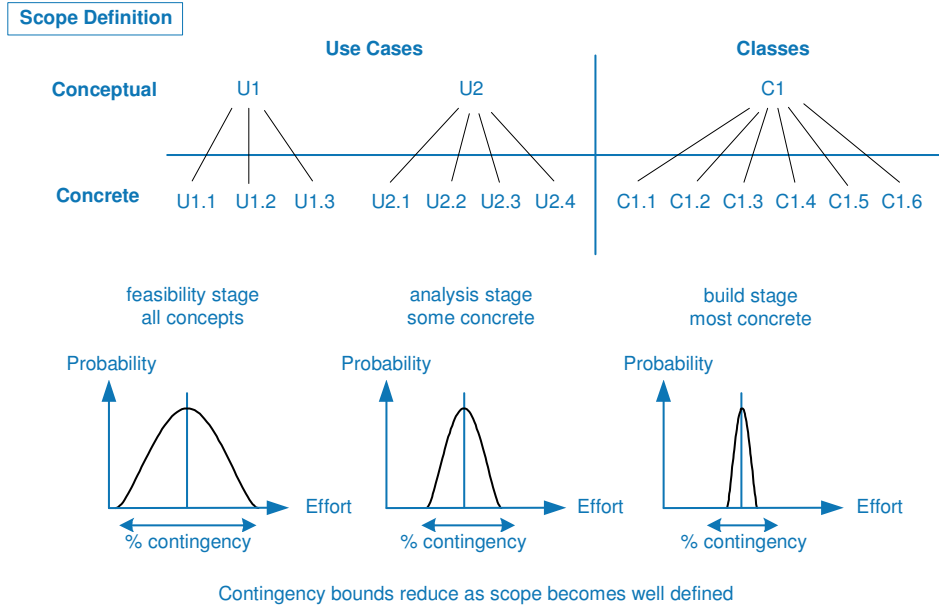
concept metrics allow for a scope population increase

Clearly the concept metrics are substantially larger than the concrete metrics as they essentially incorporate effort to construct a number of concrete software artifacts according to the population ratios.

For example, the class population ratio is 1:6 – one concept class typically results in the construction of 6 concrete classes. Therefore the concept metric is calculated as 6 times the concrete metric plus an additional discovery metric to account for the time spent in transforming the early analysis model into a detailed design model.

example

For example, the concept class 'Customer' may, through the analysis and design process, in fact become a number of implementation classes – Customer, Business Customer, Personal Customer, Business Customer Screen, Personal Customer Screen, and Persistent Customer. The concept metric allows for this likely population increase. To estimate the full implementation of this concept class we add the discovery metric of 12 hours to 6 x concrete metric of 8 hours (given the predicted population increase) to obtain an overall effort of 60 hours.



add contingency during the analysis stage

If software artifacts are all defined as high-level analysis concepts, it is advisable to add a percentage contingency to the estimates. This contingency level can be reduced as the project progresses into design and concrete software artifacts are discovered.

What is an activity profile?

a full lifecycle estimate

In isolation these productivity metric values may seem high but remember that this is effort distributed across all of the various development activities in a typical iterative software development process such as the Rational Unified Process (RUP). We are assuming that projects will follow a complete software development lifecycle to produce well-engineered software artifacts.

built-in activity profiles

ObjectMetric defines an activity profile for each software classifier at each stage in the lifecycle. In Circa the default activity profiles are as follows:

Concept Metrics

% effort	use cases	classes	subsystems	components	interfaces	web pages	scripts
planning	15	5	15	10	10	10	5
analysis	25	15	20	15	20	15	5
design	15	20	15	20	20	20	15
build	10	35	5	10	15	20	50
testing	20	10	15	20	20	10	15
integration	10	10	20	15	10	15	5
review	5	5	10	10	5	10	5

Concrete Metrics

% effort	use cases	classes	subsystems	components	interfaces	web pages	scripts
planning	6	2	7	5	4	5	2
analysis	13	6	10	7	9	9	2
design	19	21	18	23	23	23	13
build	15	44	7	14	20	25	58
testing	30	13	23	27	27	12	17
integration	15	12	30	20	14	19	6
review	2	2	5	4	3	7	2

divides the productivity metric into effort for each development activity

The activity profile divides the productivity metric into effort for each development activity. The activity profile is defined as a set of seven percentages that add up to 100%. The relative proportion of effort in each activity will vary according to the software classifier and the stage in the lifecycle.

For example, effort to develop use cases is typically biased to analysis, design and testing, whereas effort to develop classes reflects more focus on design and build. Likewise, there will be a higher proportion of time spent on planning and requirements early in the lifecycle in contrast to the emphasis on build, integration and testing later in the lifecycle.

example

The activity profiles allow the overall estimate of effort (defined by the productivity metric) to be broken down by development activity. For example, 35% of the effort in developing a concept class is allocated to build – this means that 60 hours x 35% = 21 hours is attributed to coding for each class. Likewise, 25% of the effort in developing a concept use case is allocated to analysis – this means that 90 hours x 25% = 22.5 hours is spent in capturing and documenting user requirements for each use case.

What part do qualifiers play?

a refinement to the estimate

Qualifiers provide a refinement to the estimate. Qualifiers permit a more subjective decision on the size, complexity, reuse and genericity of each software artifact.

size reflects the quantity of work to be done

Size is simply a function of the quantity of work to be done or the scale of the task to be undertaken. The bigger a task the longer it takes to complete, whether or not the activity is complex. Even a highly repetitive and straightforward task can be time-consuming if there is a substantial quantity of processing involved. ObjectMetrix defines five levels for the size qualifier: tiny, small, medium, large and huge.

complexity indicates the degree of algorithmic content

Complexity depends on the degree of difficulty. High complexity indicates a high degree of diversity, numerous inter-relationships, significant algorithmic content and/or many decision points. As complexity increases, development effort tends to increase. ObjectMetrix defines five levels for the complexity qualifier: trivial, simple, medium, difficult and complex.

reuse reflects a benefit from pre-existing software

An additional factor, not often adequately accounted for in other estimation techniques, is software reuse. Reuse means benefiting from an earlier investment. A high reuse level indicates extensive use of pre-existing software artifacts perhaps from an off-the-shelf class library or existing software infrastructure. As reuse increases development effort tends to decrease. ObjectMetrix defines four levels for the reuse qualifier: none, low, medium, and high.

*genericity
indicates design
of software for
reuse*

Reusable software can influence an estimate in two ways. As well as reuse of existing software, a project can have the primary objective of developing software *for* reuse. A high level of genericity indicates that the software artifact is required to be very general purpose, well documented, highly reliable and efficient to satisfy the requirements of numerous potential clients. As genericity increases development effort tends to increase. ObjectMetrix defines four levels for the genericity qualifier: none, low, medium, and high.

What is the impact of qualifiers?

*reduce or
increase effort*

ObjectMetrix defines levels for each of the qualifiers. The effort associated with each software artifact is adjusted by a qualifier delta to reflect the levels selected. If medium is selected there is no effect on the productivity metric. Otherwise the productivity metric is either reduced or increased by a small percentage.

*applied
cumulatively*

The qualifier algorithms in ObjectMetrix model the impact of multiple qualifier values. Rather than simply aggregating the effects of qualifier values, the effects are compounded to produce an interrelated cumulative result.

*large and
complex things
take longer*

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 – to reflect the likelihood of very large, highly complex, and extremely generic solutions taking substantially more effort to construct.

*small and simple
things take less
time*

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 – even the smallest, most trivial software artifact with high reuse will still require some minimal effort to construct.

How can I reflect the impact of various technologies?

*impact of
programming
language or
environment*

A technology choice can be made that affects the productivity of software development. Typically this is the choice of programming language or development environment.

*a percentage
impact on
certain
development
activities*

A technology defines a percentage impact on specific development activities. For example, a choice of programming language may impact build, integration and testing depending on the richness of the built-in class libraries and build environment. Likewise, the choice of a good CASE tool should have a positive impact on the analysis, design and review activities.

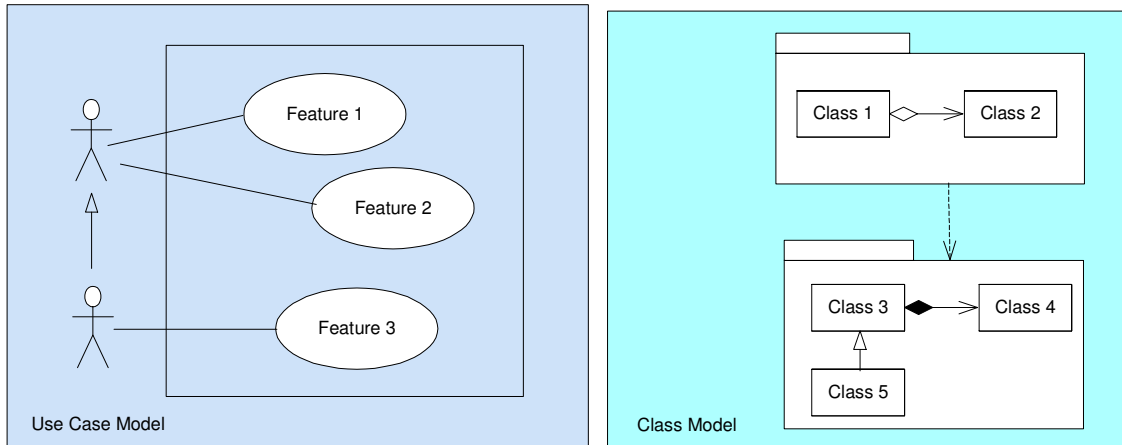
*technology
neutral estimate*

Alternatively, the technologies can be left unspecified. In this case an estimate is calculated that is independent of specific technology, where the choice of technology has not yet been made. The technology independent estimate accounts for the intrinsic effort related to software construction irrespective of technology choice.

How do I identify the project scope?

an approximation of project scope will result in a useful first-cut estimate

At the outset of a project, when the requirements are being explored, it can be a challenge to accurately identify everything that may eventually be 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 analysis takes place, and the detail is fleshed out, an increasingly accurate result can be obtained.



express requirements as use cases

Typically, most organisations start the scoping exercise by gathering user requirements, and expressing these as use cases. Use cases define discrete chunks of end user functionality – geared to cover a complete end user task and deliver business benefit for the user.

extract key concepts – subsystems, components, interfaces and classes

From this functional view, software engineers then start to analyse the key business concepts involved to establish the software architecture – the subsystems, components, interfaces and key classes that form the high-level structural design for the system. For a web-based solution, the architecture may be expressed in terms of web pages and scripts.

Can I estimate from use cases alone?

no proven correlation between the number of use cases and the number of classes

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 high functionality biased, to back-end engines which provide a limited interface but implement complex calculations using a large number of specialist classes.

business information can be obtained from a one-line description of each use case

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.

How do we calculate software effort?

metrics are added together to calculate a total project effort

The metrics are cumulative which means that they are added together to calculate a total project effort. Effort for each class allows time to define class attributes, operations and relationships, develop class diagrams and state models, code, integrate and perform unit tests. The effort for each use case takes account of the analysis of end user functionality, designing the collaborations and interactions between classes, developing controller classes or specialist functionality within existing classes, and testing the system through use case scenarios. Effort associated with each subsystem is the additional effort to package the functionality as a separate application, time to get the user interface and database schema consistent, create configuration files and perform system tests. All of this effort is added up. We have found that the most reliable estimate of effort is a factor of both structure (subsystems and classes) and behaviour (use cases).

raw effort = number of software artifacts x productivity metric

Raw effort is calculated by counting the number of software artifacts of each type and multiplying this by the appropriate productivity metric for that software classifier. In other words the total effort for classes is the number of classes multiplied by the class productivity metric.

If there were 10 concept classes the raw effort would be 10 x 60 hours = 600 hours.

partition by activity profile

The raw effort is then partitioned according to the percentage breakdown in the activity profile for the relevant software classifier into seven activity efforts.

For the 10 classes above the class activity profile indicates the following: 5% in planning = 30 hours, 15% in analysis = 90 hours, 20% in design = 120 hours, 35% in build = 210 hours, 10% in testing = 60 hours, 10% in integration = 60 hours and 5% in review = 30 hours.

adjust by qualifier delta

The activity efforts are then adjusted by the qualifier deltas in accordance with the percentage of each classifier falling into each qualifier level. In other words, if 50% of the classes are tiny and the other 50% of the classes are small, then the effort for classes is reduced by a small percentage.

The impact of this size qualifier setting on our 10 classes above is to reduce the effort by a small percentage delta. A 10% reduction for the tiny classes and a 5% reduction for the small classes - 7.5% impact overall. Total effort is reduced from 600 hours to 555 hours. If all qualifier settings are all medium this step can be skipped.

adjust by technology impact

The qualified efforts are then adjusted to reflect the percentage impact of the technology choice. Technology impacts specific development activities by a given user-defined percentage.

A choice of programming language may have a 10% positive impact on build (reducing the effort in the build activity by 10%). This would adjust the effort for our 10 classes by reducing the effort in build from 194.3 hours to 174.8 hours. Total effort is further reduced to 535.6 hours. If no technologies have been applied this step can be skipped.

sum effort for all classifiers

The total project effort is then the sum of the qualified and technology adjusted efforts for all classifiers.

For our 10 classes the total effort is 535.6 hours. By repeating this calculation for each of our software classifiers and summing the results we can calculate a total effort for the project.

How do we calculate duration and cost?

take account of resources

To extrapolate from effort to duration and cost, we need to take account of the resources assigned to the project. The number of resources, their skill level and their organisation into project teams are all significant factors in establishing project duration.

a team can perform more efficiently than any individual

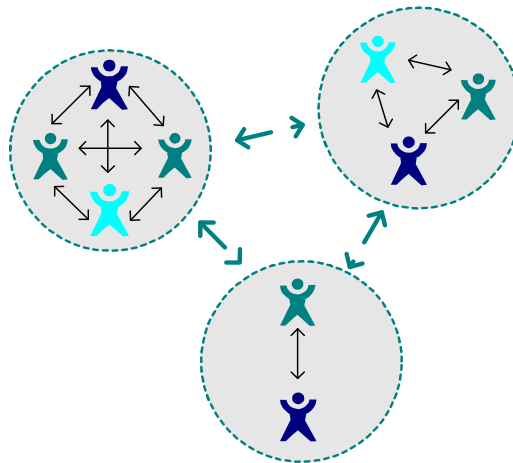
Software development remains a labour intensive activity. Where there is a certain amount of effort required, clearly two or more developers will complete the project more quickly than one. Team members can co-operate and delegate activities enabling the team to perform better and more efficiently than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

as team size increases communication and reporting overheads increase

This is true to a point, however as the team size is increased, communication levels also increase and channels of communication 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 assigning additional resources to the project will decrease duration until the inter-personal and inter-team communication overhead outweighs the advantage of further partitioning the work.

take account of duration, frequency and quality of communication

A communication overhead is calculated to take account of the duration, frequency and quality of the communication. Duration is how long the communication lasts. Frequency is how often communication occurs. Quality is the level of understanding exchanged. Duration, frequency and quality are all influenced by skill level. ObjectMetrix takes this into account by a concept termed the 'noise factor', which is derived from the number of resources, their skill levels and their team organisation. This provides a non-linear equation to calculate duration from effort.



add contingency based on risk assessment

The resulting duration can then optionally be adjusted by a contingency percentage based on an assessment of project risk (risk assessment and contingency calculations are explored in a separate technical briefing).

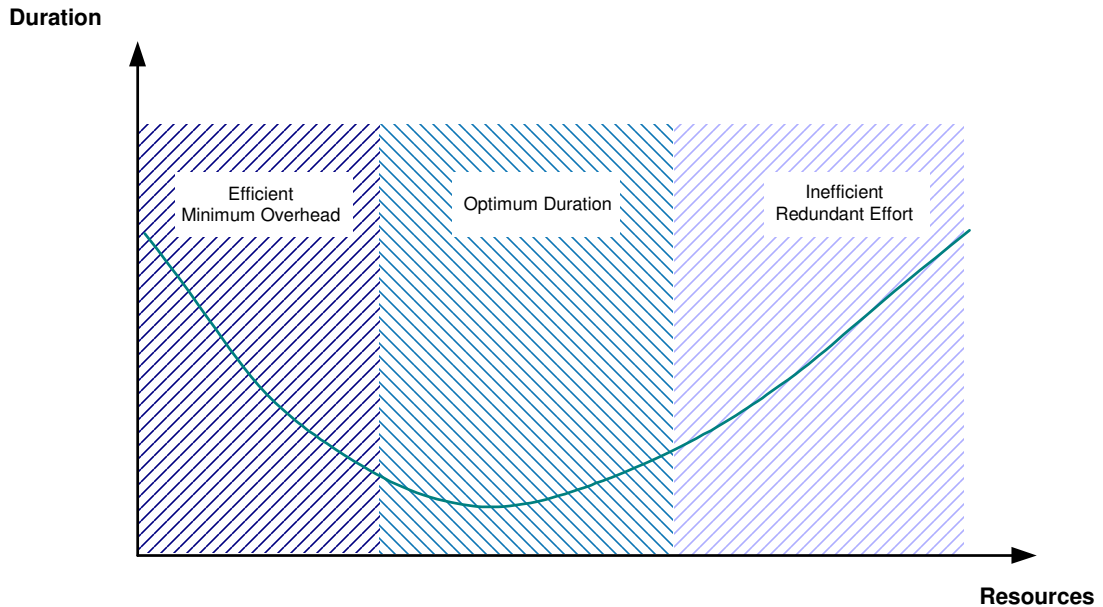
cost based on rates of pay

Cost is a simple extrapolation from person days, based on the standard and overtime rates paid to resources.

What does the optimum resource graph display?

plot resources against project duration

A graph can be plotted to map the number of resources against the project duration – this gives a good indication of the most efficient use of resources and the optimum duration.



1st region – efficient use of resources

There are essentially three regions within this graph. In the first region, the duration drops steeply as resources are added to the project. This is the region in which we achieve efficient use of resources.

2nd region – optimum duration

In the second region of the graph the duration reduces but more slowly, and eventually bottoms out and starts to increase again. This region provides the optimum duration, but typically this is offset by higher communication overheads. In other words, duration can be further reduced but at the cost of extra inter-personal and inter-team communications.

3rd region - redundancy

The third region is one of high overheads and increasing project duration, possibly to the point where an individual could complete the task more quickly than the team. This is a very inefficient allocation of resources. It typically means that there are too many resources for the amount of effort and the team could be assigned additional work. However, sometimes this situation also arises when the team is mentoring novice team members.

How do we gather actual results?

refine metrics to increase estimating accuracy

The built-in metric data forms a valuable start point. However, every organisation will have its own unique set of processes and environments. By gathering actual results from completed projects and comparing these with estimates as part of your process improvement strategy, productivity metrics can be refined to produce estimates of ever increasing accuracy.

gather actual results

The first stage in refining metric data is to gather actual results. There are three important factors in metrics gathering – volume of data, granularity of data and consistency of data.

<i>volume of data</i>	The more comprehensive the database of historic actual data, the more confidence we can place in any refinements made to the productivity or technology metrics. A small population of results is liable to be skewed by the personalities or specific skills of the engineers involved or by the specific technology and environment of the projects measured.
<i>granularity of data</i>	The granularity at which data is collected is also important. It needs to be specific enough that the comparison with estimates makes logical sense but not so detailed that the data cannot be gathered in a practical fashion.
<i>consistency of data</i>	Finally, it is essential to compare 'like with like'. This requires a consistent approach to scoping software projects and calculating estimates. If software managers do not use a consistent method of defining tasks, then comparing the effort required to complete different tasks will be meaningless.

How do we adjust metrics to reflect actual results?

<i>adjust metrics and store in a template for sharing between projects</i>	<p>By analysing the results against the estimates we can observe a number of possible outcomes:</p> <ul style="list-style-type: none"> • the estimates are generally always higher than the actuals – calculate the percentage difference and reduce the productivity metrics accordingly • the estimates are generally always lower than the actuals – calculate the percentage difference and increase the productivity metrics accordingly • the estimates are reasonably accurate but the time spent in the various activities is disproportionate with the estimates – adjust the activity profiles accordingly <p>The new metrics can be stored in a template to be used on future projects.</p>
--	---

What does it mean to drop activities from an estimate?

<i>some activities are out-sourced</i>	Another common situation is where we need to adjust the metrics to remove one or more activities. For example, we may wish to reflect that build, integration and testing is being out-sourced to another company.
<i>remove effort for non-active activities</i>	The technique that is applied here is a combination of reducing the productivity metric and adjusting the activity profile. The process is to calculate the effort for the activity to be removed, and reduce the productivity metric by this amount. The activity profile percentage for this activity is set to zero, and the previous percentage amount for the activity is distributed in proportion to the remaining activities. Circa automates this calculation according to the active switches for each development activity. These can be set for the project as a whole within the Metrics tool or set on an individual software artifact basis.