PART 1

Performance Engineering

Performance Engineering represents a cultural shift in the way organizations view their essential processes.

It embraces practices and capabilities that build quality and performance throughout an organization.

This enables organizations to increase revenue, customer attraction and retention, brand value, and competitive advantage—all while focusing on meeting and exceeding the expectations of their end users.

What Is Effective Performance Engineering?

While Performance Engineering is often defined narrowly as ensuring that nonfunctional requirements are met (such as response times, resource utilization, and throughput), the trend has moved toward a much broader application of the term.

“Performance Engineering” doesn’t refer only to a specific role. More generally, it refers to the set of skills and practices that are gradually being understood and adopted across organizations that focus on achieving higher levels of performance in technology, in the business, and for end users.

Performance Engineering embraces practices and capabilities that build in quality and performance throughout an organization, including functional requirements, security, usability, technology platform management, devices, third-party services, the cloud, and more.

Stakeholders of Performance Engineering run the gamut, including Business, Operations, Development, Testing/Quality Assurance, and End Users.

Hardware

The traditional goal of Performance Engineering between the 1970s and the late 90s was to optimize the application hardware to suit the needs of the business or, more accurately, the IT organization that provided the services to the business.

This activity was really more of a capacity planning function, and many teams charged with carrying the mantle of performance reported to operations or infrastructure teams.

Software

Over the last 30 years, software has transformed from monolithic to highly distributed, and even the concept of model-view-controller (MVC) has evolved to service-oriented (SOA) and micro-service architectures, all in an effort to reduce the number of points of change or failure, and improve the time-to-value when new functionality is implemented.

Isolating components also allows developers to test their discrete behavior, which often leaves end-to-end integrated testing out of scope.

The assumption here is that if every component behaves as it should, the entire system should perform well.

In an isolated environment this may be true, but there are many factors introduced when you're building large-scale distributed systems that impact performance and the end-user experience—factors that may not be directly attributed to software, but should be considered nonetheless, such as network latency, individual component failures, and client-side behavior.

Culture

Every organization and group has a mission and vision.

While they strive to attain these goals, performance becomes implied or implicit. But performance needs to be a part of all decisions around the steps taken to achieve a goal; it forms the basis of how an organization will embody Performance Engineering throughout their culture to achieve their mission and vision.

We need to treat performance as a design principle, similar to deciding whether to build applications using MVC or micro-services architectures

Throughout this book are sidebars in which we look at five companies and examine how performance is built in to their culture.

What is Digital Transformation?

Digital Transformation entails revamping and modernizing enterprise-level activities by leveraging digital technologies to reach out effectively and serve the user base.

It also involves gauging the impact of digital technology across the enterprise's workflow and establish ways to get services delivered with the desired impact.

Can Performance Engineering power such accuracy and resilience?

Performance Engineering and Testing with its intrinsic benefits can help you deliver the desired applications and systems in a digital set-up.

Technically, Performance Engineering is a discipline that comprises practices, processes, and methodologies, which, when applied in the software development cycle, help an application or system to deliver non-functional requirements.

Here are the five companies we will look at in more detail:

- Google
- Wegmans
- DreamWorks
- Salesforce
- Apple

Google: A Performance Culture Based on 10 Things

- 34% of U.S. adults use a smartphone as their primary means of Internet access.
- Mobile networks add a tremendous amount of latency.
- We are not our end users. The new devices and fast networks we use are not necessarily what our users are using.
- 40% of people abandon a site that takes longer than 2–3 seconds to load.

- Performance cops (developers or designers who enforce performance) is not sustainable. We need to build a performance culture.
- There is no “I” in performance. Performance culture is a team sport.
- The first step is to gather data. Look at your traffic stats, load stats and render stats to better understand the shape of your site and how visitors are using it.
- Conduct performance experiments on your site to see the impact of performance on user behavior.
- Test across devices to experience what your users are experiencing. Not testing on multiple devices can cost much more than the cost of building a device lab.
- Add performance into your build tools to automatically perform optimizations and build a dashboard of performance metrics over time. Etsy notifies developers whenever one of the metrics exceeds a performance goal.
- Surfacing your team’s performance data throughout development will improve their work.
- Celebrating performance wins both internally and externally will make your team more eager to consider performance in their work.

Google Experiment Results

- Google conducted experiments on two different page designs, one with 10 results per page and another with 30.

- The larger design page took a few hundred milliseconds longer to load, reducing search usage by 20%.

- Traffic at Google is correlated to click-through rates, and click-through rates are correlated with revenue, so the 20% reduction in traffic would have led to a 20% reduction in revenue.

- Reducing Google Maps’ 100-kilobyte page weight by almost a third increased traffic by over one-third.

Why Is Effective Performance Engineering Necessary?

Revenue

Effective Performance Engineering enables organizations to increase revenue in several ways.

In a recent survey, 68% of respondents expected Performance Engineering practices to increase revenues by "ensuring the system can process transactions within the requisite time frame."

Competitive Advantage

There's an obvious business reason to focus on the needs of end users.

If they're your customers, they're consuming your products and/or services and possibly paying you for results.

If you're a provider in a technology chain that defines a complex solution of some sort, you're still dependent on the satisfaction of users, however indirectly.

Customers: Acquisition and Retention

Acquiring and retaining customers should be the driving force in an organization, no matter the size.

As we have mentioned, Performance Engineering plays a considerable role in enabling your organization to succeed in the marketplace, and success is defined in many ways for different organizations, and is not exclusive by industry or organization or product/service.

But in general, success is defined by a few factors (and many times a combination), including: revenue, competitive advantage, brand value, and customers (acquisition and retention).

Brand Value

In many cases, businesses continue to invest in both a capability and a culture as they work to build in the practices of Performance Engineering.

These practices enable them to grow faster and become more stable.

Focusing on Business Need

The business needs to ensure that revenue, competitive advantage, customer acquisition and retention, and brand goals are achieved. Doing so means expanding products and service offerings and/or businesses either through organic or acquisition approaches, all of which may depend on an existing or new platform(s), so end users can consume products and services without interruption when, where, and how they want.

Wegmans believes in:

Caring
We care about the well-being and success of every person.

High Standards
High standards are a way of life. We pursue excellence in everything we do.

Making A Difference
We make a difference in every community we serve.

Respect
We respect and listen to our people.

Empowerment
We empower our people to make decisions that improve their work and benefit our customers and our company

Apple

1. Honest. There is high integrity in all interactions, with employees, customers, suppliers, and other stakeholders;
2. Performance-focused. Rewards, development, and other talent-management practices are in sync with the underlying drivers of performance;
3. Accountable and owner-like. Roles, responsibilities, and authority all reinforce ownership over work and results;
4. Collaborative. There's a recognition that the best ideas come from the exchange and sharing of ideas between individuals and teams;
5. Agile and adaptive. The organization is able to turn on a dime when necessary and adapt to changes in the external environment;
6. Innovative. Employees push the envelope in terms of new ways of thinking; and
7. Oriented toward winning. There is strong ambition focused on objective measures of success, either versus the competition or against some absolute standard of excellence.

The four Ws of performance engineering

Here are the four Ws to consider in performance engineering and a breakdown of how they compare to performance testing.

Why:

We use performance testing to simulate how a system will perform under production loads and to anticipate issues that might arise during heavy load conditions.

Performance engineering is driven by business requirements.

The aim is to provide better business value for the organization by discovering potential issues early in the development cycle.

The cost of fixing a bug increases in orders of magnitude as you move down the line from the design phase to production. So why wait?

What:

Performance testing simulates production loads to detect potential performance bottlenecks.

Engineers following performance engineering methodologies optimize the application for performance from the earliest design stages.

When:

Performance testing is a distinctive QA process that occurs once a round of development is completed, while performance engineering is an ongoing process that occurs through all phases of the development cycle, from the design phase, to development, to QA.

Who:

Dedicated QA teams execute performance testing.

In contrast, because performance engineering occurs throughout the development cycle, everyone takes part.

This approach ensures that by the time the system gets to QA, it's already in an optimized state.

The responsibility for performance starts with software designers and system architects, extends to the developers who do the coding, and ends with QA.

Performance engineering produces great return on investment for two reasons: It reduces the need to rework and refactor the application in later development cycles, and it results in an application that performs better precisely because performance was an early consideration and an integral part of the design.

As a result of the move to performance engineering, the classic roles that different engineers play in the software development cycle have blurred. Whether you are a designer, a coder, or a QA tester, system performance is now your responsibility.

Performance categories

Application performance is not just limited to a software or hardware perspective.

As applications become more complex and platform agnostic, there a number of performance factors that need to be taken into consideration, right from the start of the application development process:

- **Speed (Response Time)**: The amount of time the system takes to respond to a user request or internally with its own modules.

- **Scalability**: The ability of an application to quickly adapt to increasing workload by adding resources accordingly. The two types of application scaling are horizontal and vertical.
- **Stability & Resiliency**: The ability of an application to recover from certain error types and at the same time, remain functional from the user perspective.
- **Application performance degradation**: The application performance degrades over a period of time. (Memory leak).

The performance engineering process involves the following phases:

- **Performance Modeling** – Performance estimation of a new system, the impact of change on an existing system, or impact of workload change on an existing system.
- **Performance Prototyping** – Automatic generation and deployment of components, which mirror the predetermined behavior of actual components under design, into actual IT infrastructure and environments.
- **Performance Testing** – Software testing technique to ascertain the non-functional parameters of a system, such as responsiveness, stability, reliability, and resource usage, under different workloads.
- **Performance Analysis** – Analysis of performance test results, segregation of relevant data points, trends identification, etc to identify the root cause of performance issues.
- **Performance Tuning** – Changes are made to the application to ensure improved performance through code optimization, configuration optimization, distributed computing, and so on.

Top 10 performance engineering techniques that work

Performance and load tests produce a sea of data that can be overwhelming to analyze. Fortunately, there are a few methodical practices you can use to do this efficiently.

1. Identify tier-based engineering transactions

In the typical performance test harness, load scripts contain transactions or ordered API calls that represent a user workflow.

If you are creating a performance harness for an IoT application, the script will contain transactions and logic/behaviors representing a device.

Every deployment is unique, but here are some examples of the tiers and problems you may encounter:

- **Web tier:** A transaction that GETs a static non-cached file.
- **App tier:** A transaction that executes a method and creates objects but stops there and does not go to the database tier.
- **Database tier:** A transaction that requires a query from the database.

2. Monitored KPIs

Front-end KPIs show the current capacity by correlating user load, TPS, response time, and error rate.

Monitored KPIs tell the entire story of why an application starts to degrade at a certain workload level.

Hit rates and free resources are two illuminating KPIs for every hardware or software server.

Here are examples of hit rates you can monitor:

- Operating system: TCP connection rate
- Web server: Requests per second
- Messaging: Enqueue/dequeue count
- Database: Queries per second

free resources you can monitor:

- OS: CPU average idle
- Web server: Waiting requests
- App server: Free worker threads
- Messaging: Enqueue/dequeue wait time
- Database: Free connections in thread pool

3. Reduce the number of transactions you analyze

All of these business transactions are using shared resources of the deployment, so pick just a few to avoid analysis paralysis.

But which ones? That depends on the characteristics of your application.

The number of engineering transactions depends on how many tiers there are in the deployment: Five tiers equals five engineering transactions.

4. Wait for the test to complete before analyzing

All a performance engineer can do at this point is to calmly explain the value of running performance tests prior to going live and that the tests are intended only to verify that the app is executing as planned.

It's not the time or the place to analyze.

5. Ensure reproducible results

Run the same load test three times to completion.
For these three test executions, do not tweak or change anything within your performance test harness: not the runtime settings, not the code in the load scripts, not the duration of the test, not the ramp schedule, and absolutely not the target web application environment.

The "magic of three" will save you a ton of wasted hours chasing red herrings. It will reduce the data you need to analyze by removing irreproducible results.

6. Ramp up your load

Run ghost tests

Begin by running ghost tests, which check the system without executing load scripts.

A ghost test has no real user activity, but it is important: The system is left alone to do housekeeping.

Move to single-user load tests

Assign a single user to execute every single-user and engineering script, and start all of your tests at once.

If you have 23 scripts, you should have 23 users executing. Remember: Three times assures reproducible results.

Create concurrent user load scenarios

Move on to your concurrent test scenarios: Create a slow-ramping staircase scenario that allows for the capturing of three monitored KPI values for each set load.

In other words, configure the slow ramp of users to sustain a duration before adding the next set of users.

Your goal is to capture at least three KPI metric values during the duration of the sustained load.

For example, If you are ramping by 10 or 100 users at a time and collecting KPIs at 15-second intervals, then run each set load for a minimum of 45 seconds before ramping to the next one.

Yes, this elongates the test (by slowing the ramp) but the results are much easier to interpret. Use that magic number three again. It excludes anomalies. A spiking KPI metric that isn't sustained isn't a trend.

7. Use visualization to spot anomalies

user load increases, you should see an increase in the web server's requests per second, a dip in the web server machine's CPU idle, an increase in the app server's active sessions, a decrease in free worker threads, a decrease in the app server's operating system CPU idle, a decrease in free database thread pool connections, an increase in database queries per second, a decrease in the database machine's CPU idle, and so on—you get the picture.

8. Look for KPI trends and plateaus to identify bottlenecks

Resources are reused or freed (as with JVM garbage collection or thread pools), there will be dips and rises in KPI values.

Concentrate on the trends of the values, and don't get caught up on the deviations. Use your analytical eye to determine the trend.

You have already proved that each of your KPIs tracks with the increase in workload, so you should not be worried about chasing red herrings here. Just concentrate on the bigger picture—the trends.

9. Don't lose sight of engineering transactions

Remember those engineering transaction scripts? These can be gold mines for uncovering scalability issues.

Sometimes, if I don't have back-end monitoring, I'll rely on the data from these scripts alone. But together with monitoring, they tell a very accurate performance story.

The engineering transactions are executing on a sampling rate, so graph them in correlation with the user load. I usually name my transactions according to the tier that they reach. For example, "WEB," "APP," "MESSAGING," "DB."

10. Increase granularity for better clarity

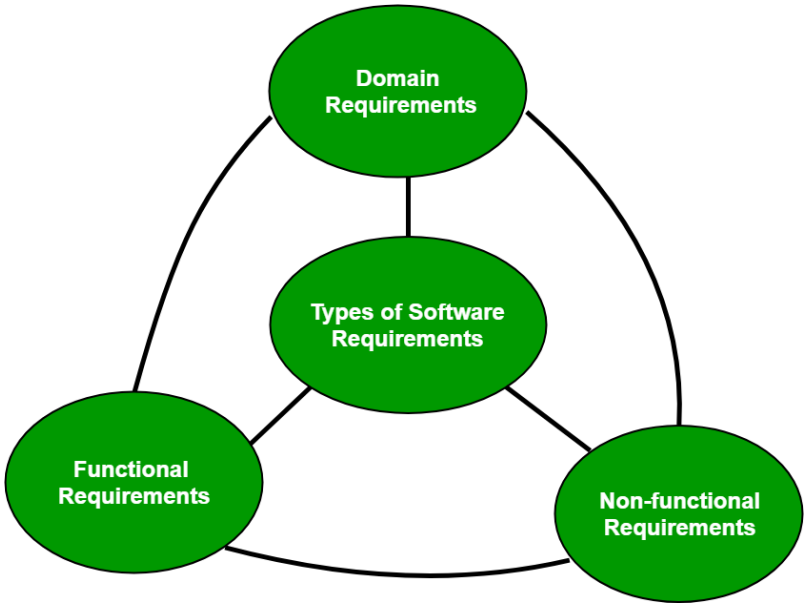
Granularity is vital for both KPI monitoring and visualization analysis. Often, if a test runs very long, the load tool will use a higher-granularity interval when graphing the results.

In effect, the tools are aggregating data and presenting only averaged sampling in graphs.

PART 2

A software requirement can be of 3 types:

- Functional requirements
- Non-functional requirements
- Domain requirements



Functional Requirements: These are the requirements that the end user specifically demands as basic facilities that the system should offer.

All these functionalities need to be necessarily incorporated into the system as a part of the contract.

These are represented or stated in the form of input to be given to the system, the operation performed and the output expected.

They are basically the requirements stated by the user which one can see directly in the final product, unlike the non-functional requirements.

Non-functional requirements: These are basically the quality constraints that the system must satisfy according to the project contract. The priority or extent to which these factors are implemented varies from one project to other. They are also called non-behavioral requirements. They basically deal with issues like:

- Portability
- Security
- Maintainability
- Reliability
- Scalability
- Performance
- Reusability
- Flexibility

NFR's are classified into following types:

- Interface constraints
- Performance constraints: response time, security, storage space, etc.
- Operating constraints
- Life cycle constraints: mantainability, portability, etc.
- Economic constraints

- **Domain requirements:** Domain requirements are the requirements which are characteristic of a particular category or domain of projects. The basic functions that a system of a specific domain must necessarily exhibit come under this category. For instance, in an academic software that maintains records of a school or college, the functionality of being able to access the list of faculty and list of students of each grade is

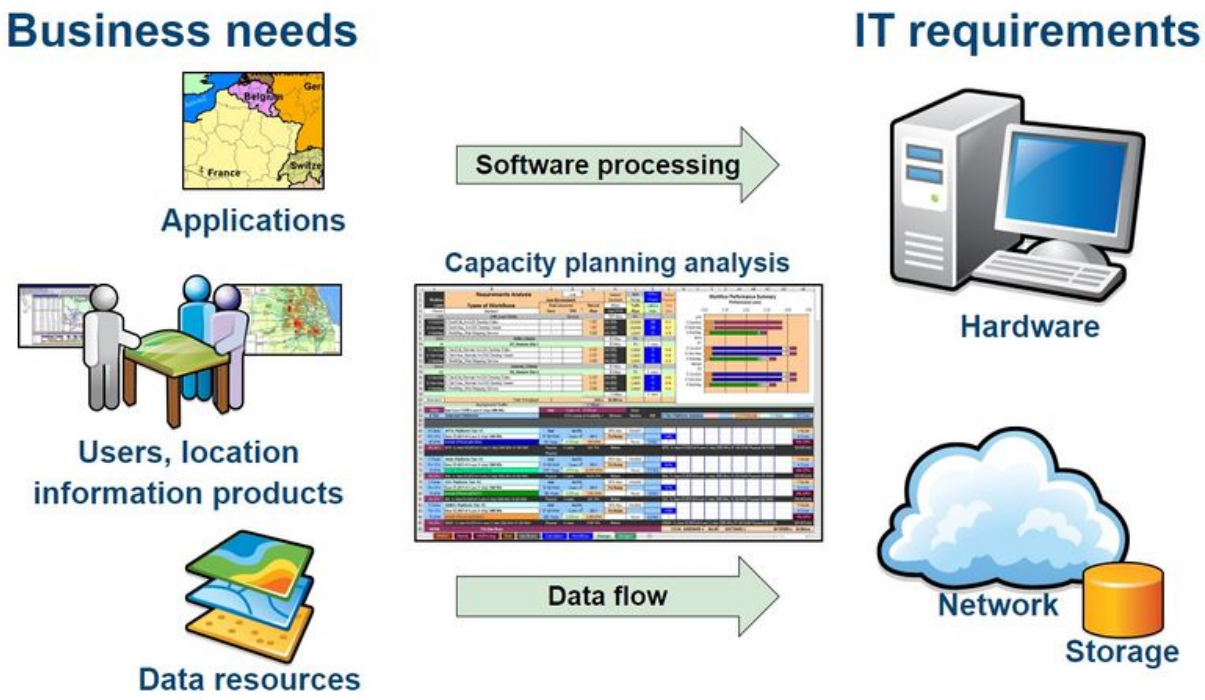
a domain requirement. These requirements are therefore identified from that domain model and are not user specific.

System Architecture Design

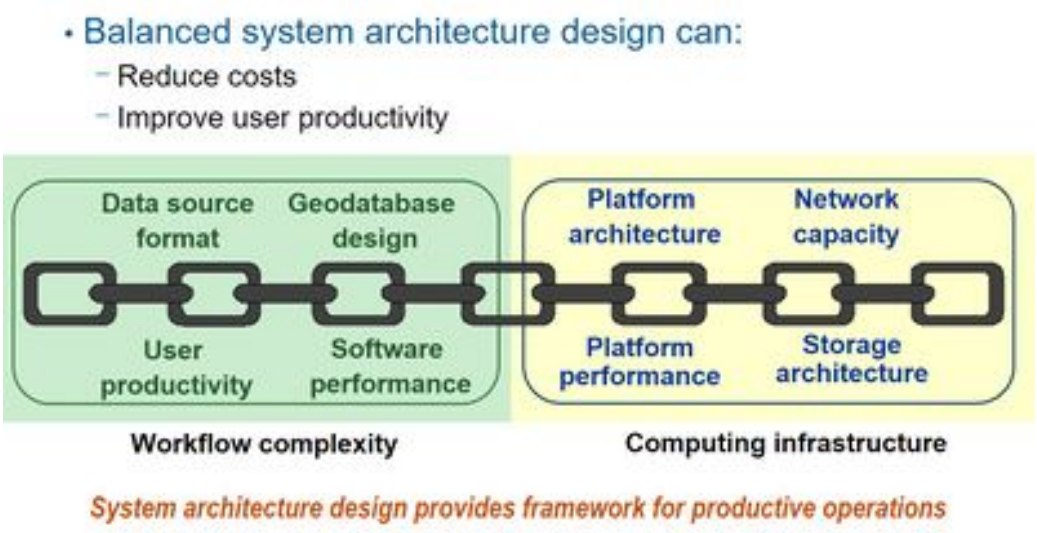
This process builds on your existing information technology (IT) infrastructure and provides specific recommendations for hardware and network solutions based on existing and projected business (user) needs.

The system architecture design process aligns identified business requirements (user needs) derived from business strategy, goals, and drivers (business processes) with identified business information systems infrastructure technology (network and platform) recommendations. The computer display transaction is the work unit used to translate business requirements to associated server and network loads. Display service times (software processing) generates platform loads and display traffic generates network loads. Peak throughput loads (business needs) are used to generate platform capacity and network bandwidth requirements.

System design starts with identifying business needs. This includes identifying user locations and required information products, identifying required data resources, and developing appropriate software applications to do the work. Business needs are represented by project workflows that identify the traffic and processing associated with each display transaction.



System performance is limited by the weakest link in the system design. System architecture design identifies the weak links during the planning process and promotes investment in a balanced system design.



Complexity is imposed on the computing architecture by the following design attributes:

- Database design and data format: DBMS, geodatabase, and ArcSDE
- User workflow software design: Application development

The computing infrastructure must provide sufficient capacity to handle peak operational loads.

- Server platform processor core and deployment architecture must handle peak processing loads.
- Network bandwidth and remote site connectivity must be adequate to avoid traffic contention.
- Storage access performance and capacity must be adequate to provide required data access.

Server performance, network capacity, and efficient storage strategies can improve user productivity and reduce system cost.

Why we do planning

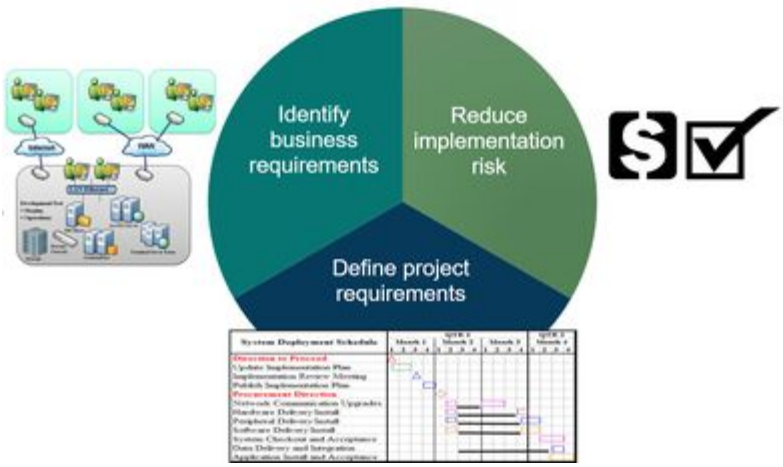


Figure 1.3 Planning is required to quantify business requirements.

The primary reasons for planning include identifying business needs, defining project requirements, and reducing implementation risk. In practical terms, we need a plan if we hope to get something done. The plan provides a foundation for successful implementation.

Business requirements may include a variety of user workflows and Web services that would improve business operations – these workflows and services must be identified early during the planning process. The workflow and service definitions establish a foundation for the rest of the planning.

The peak user workflow and service processing requirements must be identified and translated to appropriate system design loads – server processing times and network traffic. These processing loads provide a foundation for generating appropriate hardware specifications (performance and capacity requirements) and network infrastructure needs.

A project must be defined to procure and implement the required technology. The project will include a budget and schedule along with a variety of project performance milestones. Project approval is required before any major design and/or implementation efforts begin. Project approval is based on identified or perceived benefits that will accrue to the business resulting from the proposed level of time (schedule) and financial (budget) investment.

A good plan can be used to reduce system implementation risk. Delivering a system that will satisfy identified business requirements is fundamental to project success. Understanding key system performance parameters, identifying incremental system performance targets, and establishing a system performance validation plan can chart a solid framework for managing implementation risk.

Planning is an incremental process driven by rapidly changing technology. The most effective plans are designed to adapt to changing business needs and evolving technology opportunities. The established change control process must be agile and effective to ensure project tasks are managed to deliver within established schedule and budget constraints.

questions are we trying to answer?

short list of some of the more common questions:

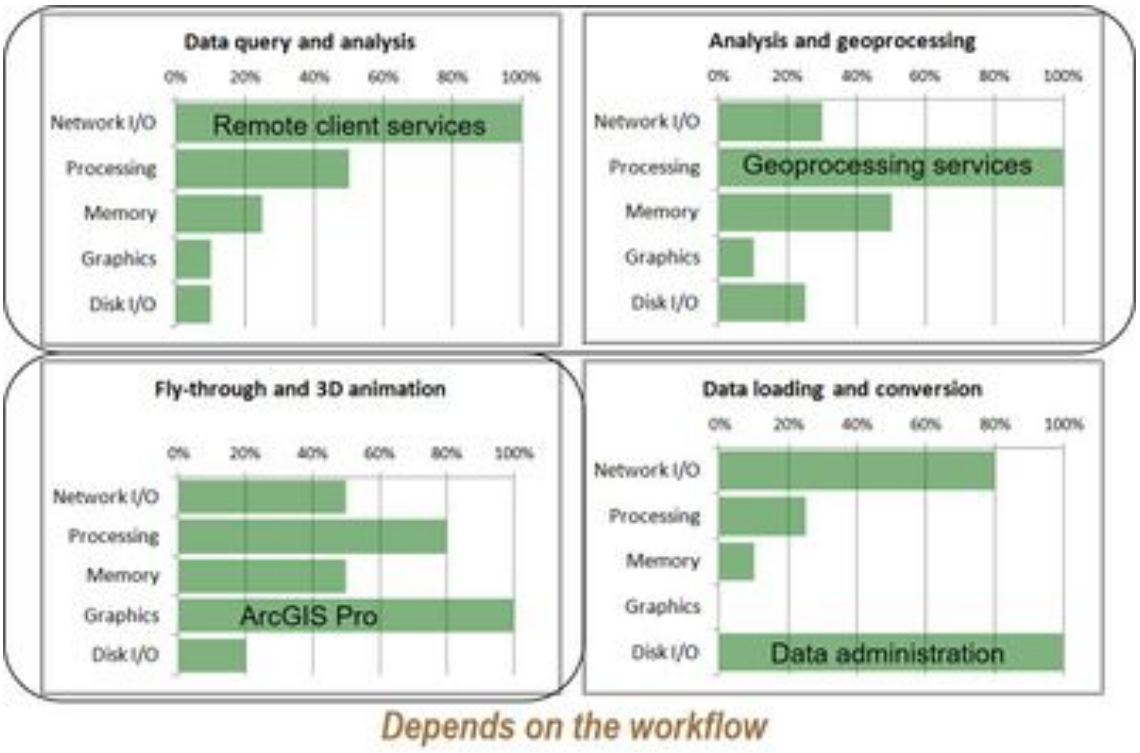
- How many users can I support with my existing hardware?
- What hardware do I need to purchase?
- How many servers (cores) do I need?
- What are the software licensing requirements?
- What workflow loads should I use for my existing applications?
- What are my current workflow service times?
- What is the capacity of my current system?

Several key infrastructure components work together to service business processing loads.

- Network I/O: Network traffic demands on available bandwidth
- Processing: Platform processor core processing demands
- Memory: Platform physical memory utilization
- Graphics: Video graphics card loads
- Disk I/O: Storage disk access performance

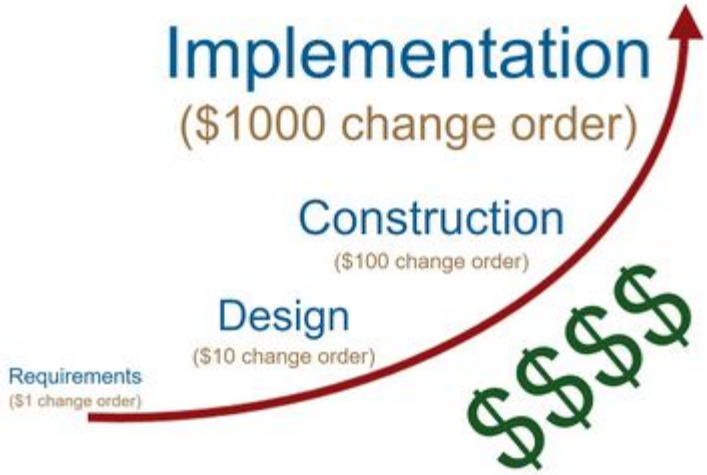
Different workflows place different loads on the system. Relative processing demands for the four use patterns above show how these loads are distributed differently, based on the workflow.

- Data query and analysis is network I/O intensive.
- Analysis and processing is compute (processing) intensive.
- Fly-through and 3D animation can be graphics intensive.
- Data loading and conversion can be disk I/O intensive.



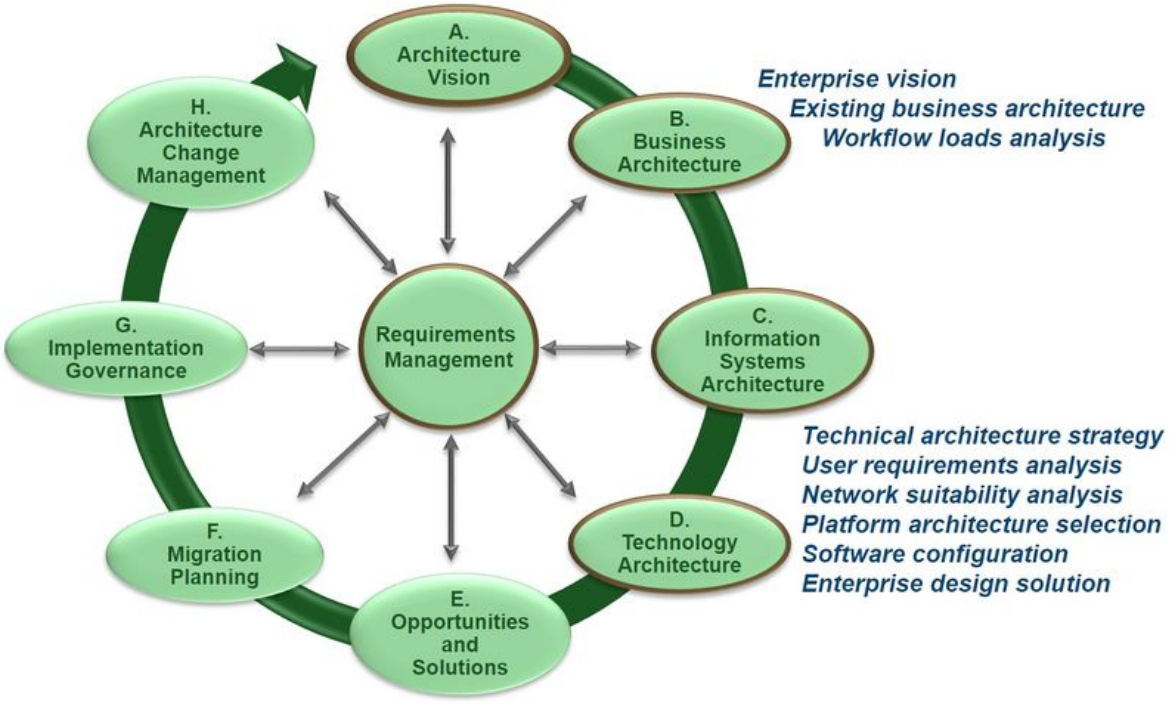
-

Cost of a change

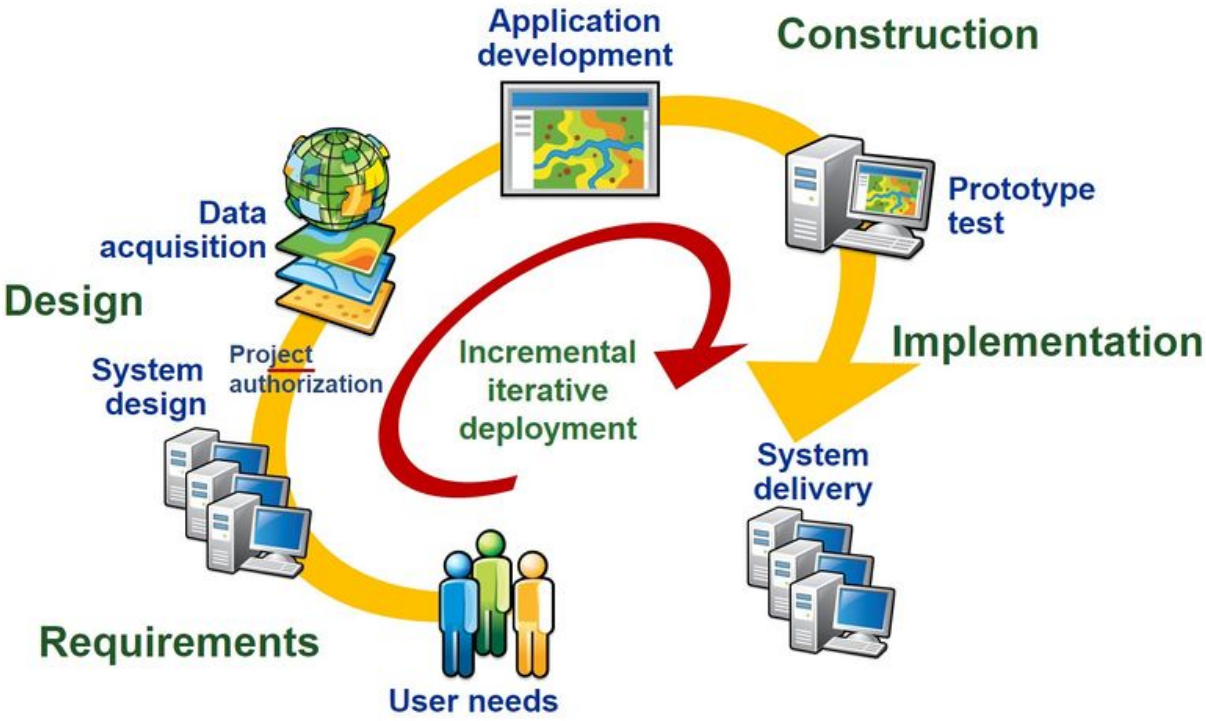


Getting it right from the start is best done by taking the time to understand the technology, quantify user requirements, select the right software technology, and deploy the right hardware. Not getting it right from the start will cost money to make it right later.

Enterprise business needs assessment



Implementation strategy



Best Practice: Deployment process is repeated incrementally on a periodic schedule to leverage technology change.

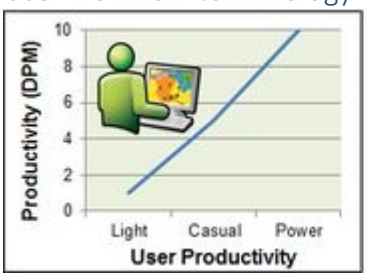
The Capacity Planning Tools (CPT) were developed as a framework to promote successful GIS system design and implementation. CPT functions contribute throughout the implementation cycle. The CPT tasks include a review of business needs, establish performance targets, identify user locations, review network suitability, select the product architecture, select the optimum software configuration, and complete the system architecture design analysis. Additional tools are provided to validate system performance during the design, construction, and implementation phases. CPT models can be easily updated and maintained to reflect changes in business requirements and review alternative system deployment strategies, providing an adaptive model for addressing a variety of incremental planning activities.

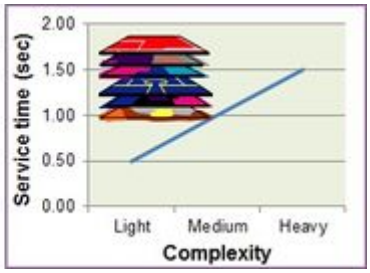
Best Practice: Build and maintain a simple system performance model that links user requirements with system design.

Capacity planning terminology

There is a great deal of terminology used within the System Design Strategies wiki that might be new to many readers. System architecture design crosses a variety of disciplines, including business operations, IT operations, and software development. The terminology introduced in this SDSwiki works to build a common vocabulary for communications that bring these groups together—establishing a framework for discussing and addressing system performance problems.

User workflow terminology





User ***workflow complexity*** is defined by service time. The ***user*** is the person working at the computer display. ***Service time*** is the average display refresh processing time.

There are three common types of user productivity: light, casual, and power.

Best Practice: Select the appropriate productivity for each user workflow.

General guidelines:

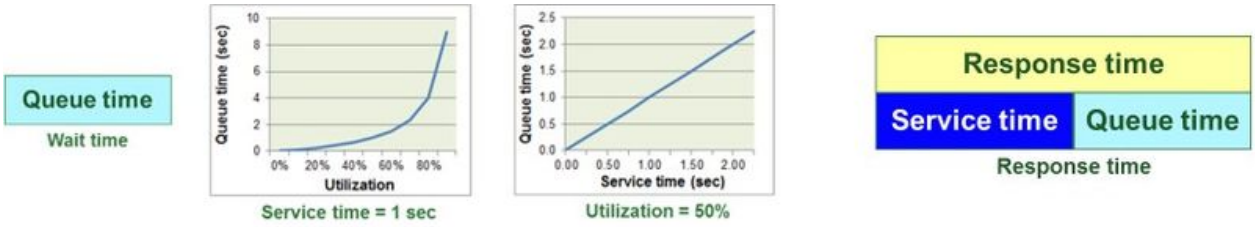
- 10 DPM is maximum user productivity for a power desktop user.
- 6 DPM is maximum user productivity for a web client workflow.
- 3-5 DPM is appropriate for casual users.
- 1 DPM is appropriate for light users.

Service time is determined by software functions, data source, and display complexity

- Light complexity is half the processing time of medium complexity.
- Medium complexity is the workflow processing baseline.
- Heavy complexity is 50 percent more than medium complexity.
- Very complex workflows are possible (up to 10x medium).

Warning: Work transaction service time directly affects system loads.
Best Practice: A heavy workflow complexity is a conservative initial planning performance target for most software technology selections.

Workflow performance



Response time is the sum of service time and queue time. ***Response time*** is the total time to complete an average display refresh. ***Queue time*** is the average process wait time due to service contention.

System queue time is predictable for large random populations.

- Large population: System throughput rates normally exceed several thousand transactions per hour, and each transaction can include hundreds of program instructions.
- Random distribution: System throughput loads are caused by hundreds of transactions generated by each user.

- Large random distribution of display transactions follows a predictable random arrival distribution.

Queue time variables include system utilization and transaction service time.

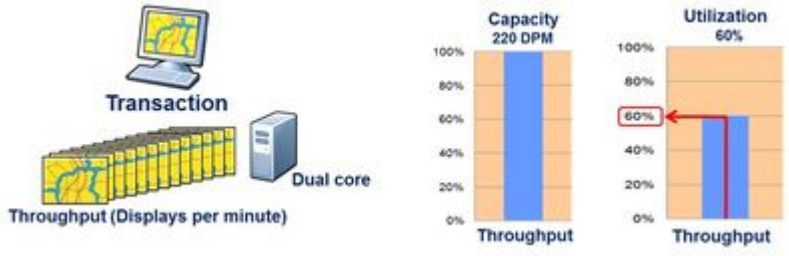
- Queue time increases with increasing platform utilization.
- Queue time increases with increasing transaction service times.

Response time is important because it can affect user productivity.

- As queue time increases, response time will increase and user productivity may decrease.
- Power users can be very sensitive to minor changes in display response times.
- User productivity can have a direct effect on business operations and the system's ability to meet user needs.

Best Practice: The faster a user can work, the more work the user can do.

System performance terminology



System performance terminology can be represented by simple relationships.

- A workflow ***transaction*** (display) is an average unit of work.
- ***Throughput*** is a measure of the transaction rate.
- ***Capacity*** is the maximum rate at which a platform can do work.
- ***Utilization*** is the percentage of capacity represented by a given throughput rate.

Display transaction is the processing load to render a new user display.

- The software program provides a set of instructions executed by the computer to complete a work transaction.
- The processor core executes the instructions defined in the computer program to complete the work transaction.

Transactions with more instructions represent more work, while transactions with fewer instructions represent less work.

Throughput is average work transactions completed over time.

- Expressed in displays per minute or transactions per hour.
- An average cumulative processing load is applied to the server.

As throughput increases, the processing load on the server increases.

Platform capacity is 100 percent throughput.

- Expressed in displays per minute or transactions per hour.
- Maximum server throughput is less than platform capacity.

Platform utilization is the percentage of time that the processor is busy.

- Processor is busy when servicing a transaction.
- Processor is not busy when waiting for a transaction.

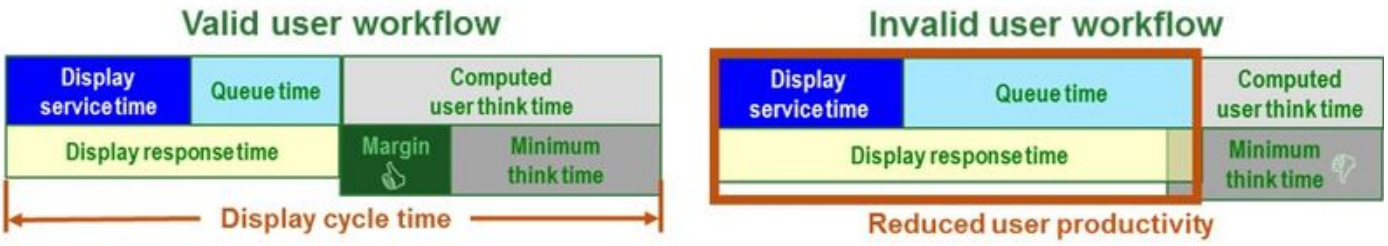
Utilization of 60 percent means that the processor is busy 60 percent of the time.

Platform utilization is the percentage of activity over a period of time.

- Short sampling periods (1 second) are difficult to evaluate.
- Longer sampling periods (30 seconds) show average utilization.
- Long sampling periods (30 minutes) may underestimate peak values.

Best Practice: Time period should represent a statistically significant sample of random transaction arrivals (100 or more transactions).

Valid workflow



What is a valid workflow?

All user workflow performance terms work together during each display transaction to satisfy business performance requirements.

Workflow specifications:

- User productivity = 10 DPM/client (user workflow performance needs)
- Display cycle time = 6 seconds (60 seconds in a minute divided by 10)

For a given display executed on a given platform:

- Display service time is a constant value.
- In a shared server environment, queue time increases with increasing user loads (increasing server utilization).
- As queue time increases, display response time increases.
- For a fixed user productivity (10 displays per minute), computed user think time will decrease with increasing display response time.

For a valid workflow, the computed user think time is greater than the minimum think time.

Warning: At some point, the computed user think time will be less than the minimum think time (invalid user workflow).

During peak system loads, the queue time can increase to a point where computed think time is less than minimum think time.

- Workflow productivity must be reduced to identify a valid workflow.
- CPT includes a RESET ADJUST function that will automatically reduce workflow productivity to the proper reduced value.

System Design Process

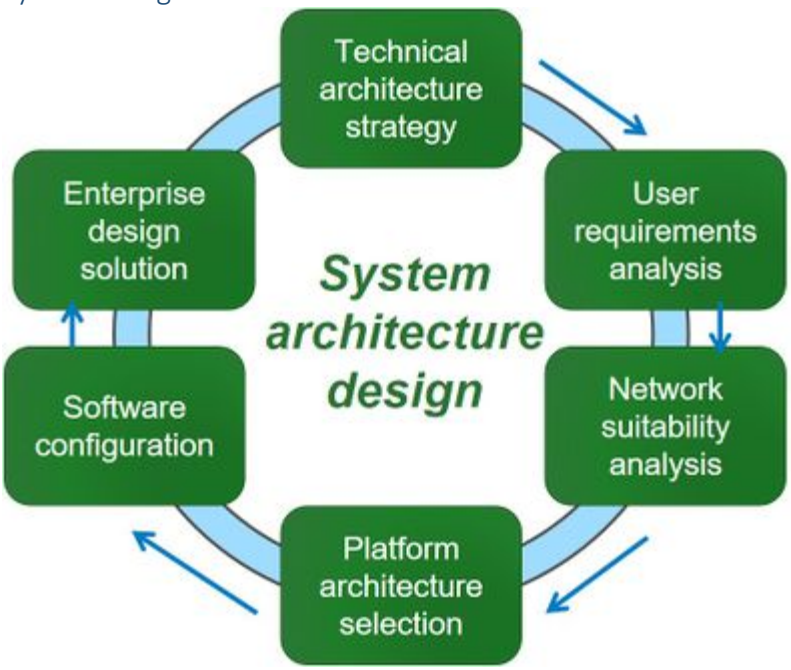
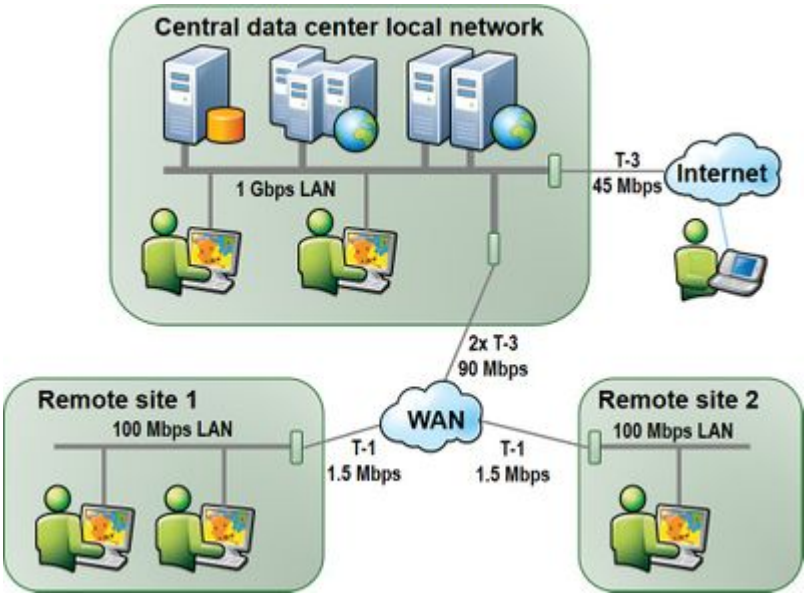
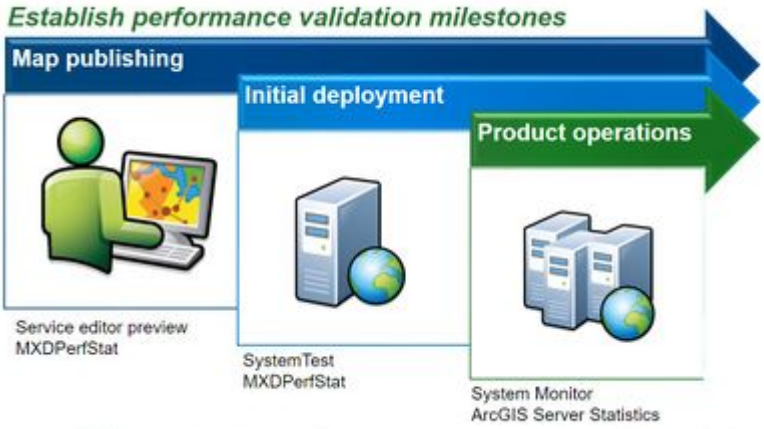


Figure 1.19 System Architecture Design process provides a logical step by step methodology for using the CPT to complete your System Architecture Design.

Once you have identified your project workflows, you are ready to complete your system design. The CPT is developed for use based on a standard system architecture design process as shown in Figure 1.19. Each cycle of the system architecture design process includes the following steps:

- Technical architecture strategy—High-level overview showing user site locations, network bandwidth connections, and central data center locations. User location information is collected during the user needs analysis.
- User requirements analysis—CPT Requirements analysis section is configured to represent the site locations, user workflows, peak loads, and network bandwidth for the enterprise design solution.
- Network suitability analysis—CPT Design completes the network suitability analysis and identifies any communication bottlenecks. Network bandwidth upgrades are identified to complete the network suitability analysis.
- Platform architecture selection—CPT Design Platform tier is configured to represent the design solution. Identify platform tier nicknames, select platforms, and identify platform rollover settings.
- Software configuration—CPT Design Software Configuration module is used to assign workflow software to supporting platform tier (software install) and make workflow data source selection.
- Enterprise design solution—Once configured, the CPT Design tab completes the system architecture design analysis and provides the platform solution.

- 

- 

CPT test tab translates performance measurements to workflow service times