



Resumo do curso - Início Rápido em Teste de Software e QA



Seção 1: Carreira em teste e QA [↗](#)

- A área de Teste e Garantia de qualidade(QA - Quality Assurance) é responsável por assegurar que

softwares e sistemas funcionem corretamente, atendendo aos requisitos e expectativas dos usuários.

O profissional de QA atua na identificação de erros (bugs), na criação e execução de testes manuais e automatizados, e na implementação de processos para melhorar a qualidade do produto.

- Teste Manuais:** Executados sem automação, verificando funcionalidades e usabilidade.
- Testes automatizados:** Utilizam scripts e ferramentas para validar funcionalidades repetitivas.
- Testes de regressão:** Garantem que novas mudanças não afetam funcionalidades já existentes.
- Testes de Performance:** Avaliam a velocidade, estabilidade e escalabilidade do sistema.

Ferramentas comuns: Selenium, JUnit, Postman, Jira, TestRail.

Habilidades essenciais: **Atenção aos detalhes, pensamento crítico, conhecimento em programação para testes automatizados e boa comunicação para relatar problemas.**

- Uma carreira em teste e QA conta com várias oportunidades em todo o mundo, com oportunidades de:

Boa remuneração

Oportunidade de evolução

Qualidade de vida

Em expansão



Segue alguns sites para procurar vagas na área:

[APinfo](#)

[Dice.com](#)

[Monster](#)

[uTest - PJ](#)

? O DEV DEVE TESTAR SEU CÓDIGO? [↗](#)

Sim, o dev deve testar seu código.

- 50% dos defeitos só o programador pode encontrar**(segundo Glenford Myers, pioneiro da qualidade).
- Exemplo: Erros de lógica interna, implementação técnica ou cálculos específicos são mais visíveis para quem escreveu o código.
- Responsabilidade compartilhada:** o professor
ênfata que **toda equipe**(devs, testadores, PO, etc.) deve colaborar nos teste.

Mas... Por que ainda precisamos de testadores?

- Viés de confirmação:** O dev
acredita no código que criou → tende a testar apenas cenários “que funcionam”. Analogia: É como reler um e-mail que você escreveu - seu cérebro “vê” o que você quis dizer, não o erro de digitação.

- **Defeitos que escapam ao dev:**
Problemas de **usabilidade**, **requisitos mal interpretados** ou **cenários não previstos** exigem um olhar externo.

A Sinergia Dev-Testador [🔗](#)

Dev	Testador
Testa para validar ("funciona como eu planejei?")	Testa para invalidar ("o que pode falhar?").
Foca em lógica interna .	Foca em comportamento externo (usuário final).
Encontra ~50% dos bugs .	Encontra os outros 50% + riscos não óbvios .

Conclusão da aula

- O professor **NÃO** disse que o dev não deve testar – pelo contrário: **é essencial que ele testa**.
- Mas testes especializados são complementares:

O que garante a qualidade técnica.
O testador garante a qualidade sob perspectivas diversas.
- Frases do professor:

"Todo mundo testa, mas o testador profissional traz técnicas para ir além do óbvio."

Soft Skills e Hard Skills do Profissional de Teste [🔗](#)

1. O Que São Soft Skills e Hard Skills? [🔗](#)

- **Soft Skills:** Habilidades comportamentais e interpessoais (ex.: comunicação, resiliência).
- **Hard Skills:** Conhecimentos técnicos (ex.: ferramentas de teste, programação).

2. Soft Skills Essenciais para um Testador [🔗](#)

Habilidade	Por Que Importa?	Exemplo Prático
Motivação	Mantém o foco em dias difíceis e projetos complexos.	"Segunda-feira, levanto com energia para aprender e corrigir erros."
Persistência	Testes e automação exigem repetição até dar certo.	"Se falhou, é porque ainda não terminei. Só acaba quando funciona."
Curiosidade	Descobre cenários inesperados e melhora a cobertura de testes.	"E se o usuário fizer X ao invés de Y? Como o sistema reagiria?"
Perfeccionismo	Busca a excelência e identifica falhas sutis.	Compara o software com o "ideal" (como um livro adaptado para filme).
Resiliência	Lida com mudanças constantes (requisitos, prazos, tecnologias).	"O projeto mudou? Adapto-me e foco na solução, não no problema."

Organização	Gerencia múltiplas tarefas, arquivos de evidências e prazos.	"Documento cada bug com prints, logs e passos para reproduzir."
Trabalho em Equipe	Colabora com devs, POs e clientes para entregar qualidade.	"Se atraso minha tarefa, impacto todo o time. Peço ajuda quando preciso."

3. Hard Skills Complementares [🔗](#)

- **Técnicas de teste** (funcionais, não-funcionais).
- **Automação de testes** (ferramentas como Selenium, Cypress).
- **Gestão de bugs** (JIRA, Bugzilla).
- **Conhecimento em SQL e análise de logs**.

4. Frases-Chave da Aula [🔗](#)

- "Soft skills fazem você ser um testador excepcional; hard skills fazem você ser contratado."
- "Testador é como um detetive: curioso, persistente e detalhista."
- "Organização salva vidas (e prazos) no mundo dos testes."

Débito Técnico [🔗](#)

Débito técnico é um conhecimento que o profissional deveria possuir, mas ainda não tem. Toda pessoa possui um débito técnico e sempre terá, o problema não é ter um débito, o problema é não planejar para apreender.


Dicas para pagar o seu débito técnico:

- Planejar
- Agir
- Rever o plano

Seção 2: Introdução ao Teste de Software [🔗](#)

Histórias do Teste de Software [🔗](#)

1. As Origens (Século XIX) – O Nascimento da Programação e dos Primeiros "Defeitos" [🔗](#)

- **Charles Babbage** projetou a Máquina Analítica (1837), considerada o primeiro conceito de computador programável.
 - Usava cartões perfurados (inspirados em teares mecânicos) para armazenar instruções.
 - Objetivo: Automatizar cálculos matemáticos complexos, substituindo os ábacos  e métodos manuais.
- **Ada Lovelace**, matemática e colaboradora de Babbage, escreveu o primeiro algoritmo para a máquina.
 - Em suas anotações, ela descreveu um possível erro nos cálculos, sendo considerada a primeira pessoa a identificar um "defeito" em software (antes mesmo dos computadores eletrônicos!).

2. A Era dos Cartões Perfurados e o Primeiro "Bug" (Final do Século XIX – Década de 1940) [🔗](#)

- **Herman Hollerith** criou máquinas de tabulação com cartões perfurados (1890), usados no censo dos EUA.
 - Esses cartões deram origem à IBM e foram a base do processamento de dados por décadas.
- **Grace Hopper** (1947) encontrou o primeiro bug real em um computador (o Harvard Mark II).

- Um inseto (mariposa) travou um relé, e ela registrou no log: "First actual case of bug being found".
- O termo "debugging" (remover erros) surgiu dessa anedota.

3. A Segunda Guerra e a Computação Moderna (Década de 1940–1950) [↗](#)

- **Alan Turing** desenvolveu a Máquina de Turing e ajudou a quebrar o código Enigma (usado pelos nazistas).
 - Seu trabalho fundamentou a ciência da computação e mostrou a importância de testar sistemas complexos.
- Os primeiros computadores eletrônicos (como o ENIAC) exigiam testes manuais demorados, pois não havia ferramentas automatizadas.

4. A Consolidação do Teste como Disciplina (Década de 1970–2000) [↗](#)

- **Glennford Myers** (1979) publicou "The Art of Software Testing", o primeiro livro dedicado ao tema.
 - Introduziu conceitos como:
 - Teste de Caixa Branca (análise de código interno).
 - Teste de Caixa Preta (focado em entradas/saídas sem ver o código).
 - Regra dos 10x: Corrigir um defeito na fase de requisitos custa 10x menos do que em produção.
- Anos 1990–2000: Surgiram as primeiras ferramentas de automação (como o Mercury Interactive, hoje HP ALM).

5. O Teste no Brasil e as Certificações (Anos 2000 em Diante) [↗](#)

- **Emerson Rios** Ricardo Cristalli e Aderson Bastos foram pioneiros no Brasil, escrevendo livros como:
 - "Base de Conhecimento em Teste de Software" (referência para a certificação CSTQB).
- ISTQB (International Software Testing Qualifications Board) se tornou o padrão global, com:
 - Mais de 1 milhão de certificações emitidas.
 - Níveis Foundation, Advanced e Expert.

6. O Teste na Era Ágil e DevOps (Década de 2010–Hoje) [↗](#)

- **Lisa Crispin e Janet Gregory** defenderam o papel do tester em metodologias ágeis, com livros como:
 - "Agile Testing: A Practical Guide for Testers and Agile Teams".
- Novas abordagens surgiram:
 - Teste Contínuo (integrado ao CI/CD).
 - IA e Machine Learning para automação inteligente.

Conclusão: Uma Jornada de Erros, Bugs e Superação [↗](#)

Desde os cartões perfurados de Babbage até os testes automatizados em nuvem, a história do teste mostra:

- ✅ **Defeitos sempre existirão** (são inerentes ao desenvolvimento).
- ✅ **Encontrar bugs cedo economiza tempo e dinheiro** (Regra dos 10x).

1. Importância do Teste X Danos dos Bugs

DANOS QUE OS BUGS SÃO CAPAZES DE GERAR		PREJUÍZOS FINANCEIROS E DE IMAGEM
EMPRESAS/ORGANIZAÇÕES	PESSOAS	
ATRASOS	CONSTRANGIMENTOS	
PERDA DE CONFIANÇA	PERDA OU SUPRESSÃO DE DIREITOS	
PERDA DE VENDA	RISCO DE VIDA E ACIDENTES	
GOVERNO	MEIO AMBIENTE	
VULNERABILIDADE DE INFORMAÇÕES	ALERTAS ATRASADOS	
DECISÕES ESTRATÉGICAS INCORRETAS	DESPERDÍCIO DE RECURSOS	
DERROTAS MILITARES	POLUIÇÃO	

O software bem testado (de maneira exaustiva) pode evitar problemas graves na vida de todos. **O testador precisa pensar no impacto daquele teste na vida das pessoas.**

- Os 7 Fundamentos do Teste (ISTQB)

1. O teste demonstra a presença de defeitos, nunca a sua ausência;
2. Teste exaustivo não é possível, exceto para casos triviais, avaliar riscos e prioridades para dar foco aos esforços de teste;
3. Teste antecipado: deve começar o mais breve possível, é necessário priorizar para que não haja prejuízo, o bug se torna mais caro a cada dia;
4. Agrupamento de defeitos: estão distribuídos de forma heterogênea, algumas partes do software terão mais defeitos do que outras partes;
5. Paradoxo do pesticida: as mesmas técnicas de testes podem não identificar novos defeitos, é necessário inovar;
6. Teste depende do contexto: um software de piloto automático de um avião deve ser testado com amplitude e profundidade diferentes de um software de um quiosque de informações em um shopping;
7. A ilusão da ausência de erros: encontrar e consertar defeitos não ajuda se o sistema construído não atende às expectativas e necessidades dos usuários/clientes.

- Diferença entre Teste e QA

O teste é focado em avaliar se o produto está de acordo com o que o cliente/usuário gostaria, se está de acordo com a documentação técnica e história de usuário etc.

O QA vai trabalhar sobre as lições aprendidas. Então não é alguém que quer melhorar o produto, ele quer melhorar o processo, a forma de fazer.

São duas coisas diferentes que se complementam.

- Erro, Ocorrência, Defeito e Falha

Softwares são feitos por pessoas para pessoas.

Pessoas cometem **erros (enganos)**, que produzem **defeitos (bugs)** no código, em um software, sistema ou documento.

Se um defeito no código for executado, o sistema falhará ao tentar fazer o que deveria (ou, em algumas vezes, o que não deveria), causando uma **falha**.

Nem todos os defeitos causam falhas, a falha só existe quando o defeito é executado. Se eu encontrar alguma coisa que fiz de errado, é um erro, se eu encontrar alguma coisa que outra pessoa fez, é um defeito.

O ideal é abordar o defeito como uma **ocorrência (uma dúvida, um incidente)**, não como um defeito.

- Tipos de Testes Baseados na IEC/ISO 25010

O que é IEC/ISO 25010? 🤖

Um padrão. Um exemplo/sugestão de como algo deveria ser feito, para que não seja necessário ficar reinventando a roda.

Os tipos de testes são:

- IEC/ISO 25010 - Adequação Funcional

É o principal teste, também chamado de teste de negócio.

O software respeita a funcionalidade? Ou seja, faz o que precisava fazer de forma completa?

O que é avaliado

Completude: o software faz uma parte do que foi solicitado ou faz tudo?

Correção: está realmente dando o resultado certo?

Apropriado: o software está entregando a informação de forma apropriada sem gerar dúvidas? Precisa fazer sentido para o público.

- IEC/ISO 25010 - Usabilidade

É a facilidade que o usuário vai ter de utilizar o programa. É algo muito intuitivo. Se subdivide em seis características:

1. Reconhecibilidade: facilitar que o usuário reconheça elementos e comportamentos (exemplo: o ícone da impressora remete à ação de imprimir – é autoexplicativo);
2. Aprendizabilidade: facilidade de aprendizado do usuário (pode ser algum tipo de assistente virtual, recursos rápidos de ajuda);
3. Operabilidade: quanto mais rápido eu conseguir realizar uma tarefa, com menos cliques ou em menos tempo, melhor será a usabilidade do programa;
4. Proteção contra erro do usuário: não permite o usuário fazer uma escolha indevida/errada (exemplo: se o usuário seleciona o país Brasil, aparecerão apenas estados do Brasil);
5. Estética (da interface do usuário): a interface deve ser visualmente agradável e ergonômica, com disposição harmoniosa de elementos, cores e informações, isso faz com que o usuário consiga acessar por mais tempo;
6. Acessibilidade: facilitar o acesso para todas as pessoas (pessoas com deficiências físicas, pessoas que não sabem ler, crianças, idosos etc).

- IEC/ISO 25010 - Compatibilidade

Coexistência: esse software é compatível com outros? Há facilidade de coexistir?

Interoperabilidade: é a facilidade de comunicar. Um software consegue se comunicar, enviar e receber dados?

- IEC/ISO 25010 - Confiança

O software sempre deve estar disponível para uso. São 4 subcaracterísticas:

1. Maturidade: prevenir a falha antes que aconteça (exemplos: impressora que avisa quando a tinta está acabando, luz do tanque reserva avisando sobre abastecimento do carro);
2. Disponibilidade: manter-se a disposição de usuários e sistemas;
3. Tolerância a falhas: perceber e compensar as falhas em tempo real (exemplo: um sistema que identifica a falta de energia e liga o gerador);
4. Recuperabilidade: recuperar-se de falhas e travamento (O que o software faz quando dá erro? Cancela? Continua? Volta do ponto que parou? Avisa o usuário?)

Precisa haver a sinalização de como lidar com isso.

- IEC/ISO 25010 - Eficiência no Desempenho

Comportamento em relação ao tempo: verificar se o software é rápido (tempo de resposta ideal é abaixo de 3 e 5 segundos);

Utilização de recursos: Como usa a memória RAM, disco, processador, rede, telecomunicação? Quais recursos ele usa muito e quais recursos usa pouco? Como aumentar a capacidade/quantidade de usuários?

É melhor um software que usa bastante recurso do que um software que não usa os recursos que tem.

Capacidade: precisa dar conta de atender todas as pessoas. Em datas especiais, temos capacidade para atender nosso público ou uma multidão de pessoas?

- IEC/ISO 25010 - Manutenibilidade

Facilidade de dar manutenção a um software. Pode ser dividida em:

1. Modularidade: é bom fazer o software por partes, porque, se der algum problema, eu consigo tirar aquela parte e encaixar uma nova, trocar por outra (exemplo: vamos imaginar que eu tenho tanta gente acessando que eu não tenho recursos o suficiente para segurar, eu poderia desligar alguma parte administrativa que não é tão relevante nesse momento e ativar mais recursos para atender os clientes numa Black Friday, por exemplo. Então, se for modular, é como se eu fosse num quadro de força e desligasse o disjuntor);
2. Reusabilidade: criar de maneira aberta e mais genérica, ou seja, quando eu vou criar algo, algum recurso, alguma funcionalidade ou alguma parte nova para um software, eu penso em criar de uma maneira que eu possa usar em outros lugares;

3. Analisabilidade: se eu ler esse código, conseguirei entender? Ou ficará a sensação de que é melhor fazer de novo? Sempre criar de uma maneira que facilite a compreensão;
4. Modificabilidade: o quão fácil é modificar esse software? Vamos supor que um componente fala com SQL Server, só que agora tem que falar com Oracle. Existe um outro componente que que mude o sistema inteiro, sem precisar mexer no programa como um todo? É como se fosse a troca de apenas uma pecinha.
5. Testabilidade: é fácil e viável testar? Eu tenho as informações, recursos e conhecimento necessários para testar? Caso haja impedimentos, é necessário falar na hora, caso haja demora, o problema virou meu. Sempre tirar dúvidas quando receber a documentação para alinhar expectativas. Há testes que oferecem riscos de acidente e não se testam ou se testam com muito cuidado. Há testes que só poderão ser feitos manualmente;

- IEC/ISO 25010 - Portabilidade

É a capacidade de um software de funcionar de forma muito próxima em vários sistemas operacionais, navegadores, smartphones, TVs, vídeo games. Ou seja, é verificar se aquele software funciona em vários equipamentos, em várias situações, em vários cenários e contextos diferentes (exemplos: um botão pode desaparecer no navegador, pode ser difícil de clicar em um botão pelo celular);

1. Adaptabilidade: facilidade de funcionar nos ambientes (as coisas precisam se ajustar com menos intervenção humana possível, exemplo: a tela se autoajusta em resolução e formato);
2. Instalabilidade: facilidade de instalar e desinstalar (exemplos: publicar, despublicar, configurar, é essa flexibilidade de ligar/desligar recursos);
3. Substituibilidade: facilidade de substituir (identificar se melhoramos ou pioramos, vamos entregar algo melhor ou pior?)

- IEC/ISO 25010 - Segurança

Um software não deve ser invadido e nem manipulado.

Confidencialidade: somente quem criou aquela informação ou está na mesma hierarquia pode acessar;

Integridade: só pessoas autorizadas podem fazer modificações ou inclusões e tem que ficar registrado quem fez e quando para evitar qualquer tipo de fraude;

Não repúdio: uma garantia que o software dá para que nenhuma das partes que comprou, vendeu

ou fez qualquer tipo de transação, possa dizer que não fez. É preciso garantir que aquela pessoa que está fazendo a transação é realmente o nosso usuário (exemplos: garantir isso através de usuário e senha, controle biométrico, controle para saber qual aparelho celular, posição do gps, qualquer tipo de conjunto de informação que indica quem é o usuário);

Responsabilidade: garantir que aquele procedimento é auditável, tem um log, se consigo ver que horas que a pessoa entrou no sistema, com que usuário, IP, dispositivo. Saber passo a passo do que ela fez, o que ela deixou de fazer. Ou seja, um recurso que permite prestar contas sobre as atividades dos usuários. Essencial em qualquer tipo de aplicação que envolve transações financeiras legais;

Autenticidade: conseguir garantir que a transação foi feita, que aquela pessoa é ela mesma, aquela compra realmente foi feita (exemplo: checagem por biometria, autenticidade de pagamento, assinar e-mail com certificado digital);

- Testes Manuais X Testes Automatizados

Foco na solução. Existem situações que não valem a pena automatizar, precisa ser avaliado.

É necessário ter a capacidade de testar tudo de novo a cada mudança do produto.

Existem muitas formas de evoluir o teste, precisa haver uma estratégia com conjunto de testes.

Testes de regressão automatizados: o objetivo não é encontrar defeito, mas sim, garantir que o que está pronto já funciona.

- Testes Tradicionais X Testes Ágeis

O testador é um membro ativo do projeto, deve participar com dúvidas e desmistificar a área.

■ Seção 3: Atitudes de um profissional da qualidade

- Pressão organizacional

Em projetos de tecnologia, prazos e recursos são limitados, gerando grande pressão sobre o profissional de testes, que precisa garantir qualidade, cumprir cronogramas e atender às expectativas do cliente. A pressão vem de múltiplos *stakeholders*.

O ideal é a comunicação com clareza e profissionalismo, mantendo a calma e a transparência sobre problemas encontrados, mesmo sob pressão. O testador atua como "conselheiro da qualidade", alertando sobre riscos de bugs e impactos de entregas prematuras, mas sem tomar decisões finais.

A entropia (tendência natural ao caos) favorece o testador: quando alertas são ignorados e problemas ocorrem, a credibilidade dele cresce. Seu papel é educar o time sobre qualidade, equilibrar demandas e promover uma cultura de testes, mostrando que **defeitos podem ter custos altos** (financeiros, reputacionais ou até humanos). **Tempo e comunicação estratégica são aliados** para transformar pressão em colaboração.

Resumo dos pontos-chave: pressão multifonte, comunicação, entropia e papel do testador como conselheiro/educador, responsabilidade e postura na entrega do software.

- Comprometido X Envolvido

Comprometido: assume responsabilidades, busca soluções e pensa no todo (visão de dono).

Envolvido: faz só o necessário, age com passividade ("não é meu problema").

1. Consequências: profissionais apenas envolvidos são os primeiros a serem cortados em crises, pois não agregam valor real;
2. Meritocracia: empresas valorizam quem entrega resultados, mas é preciso escolher ambientes que reconheçam esforços. Se não houver reconhecimento, mude de empresa;
3. Faça o que ama: comprometimento exige paixão pela área. Se não gosta de testes, dificilmente se destacará;
4. Ciclo virtuoso: quem faz bem feito é reconhecido, ganha oportunidades e melhora continuamente.
5. Visão de dono: trate projetos como se fossem seus, seja proativo, economize recursos e busque eficiência;
6. Ser antes de ter: para ser promovido a "testador sênior" ou "líder", aja como um *antes* de receber o cargo;
7. Colecione experiências: mesmo em empregos não ideais, acumule projetos e aprendizados para seu currículo;
8. Sem garantias: empresas mudam, então foque no que você controla: seu desempenho e capacidade de adaptação;

Resumo dos pontos-chave: comprometimento é sobre ação, responsabilidade e paixão, não apenas estar presente, mas fazer a diferença.

- Autogerenciamento

1. Priorize seu tempo: tempo é finito e não renovável, equilibre trabalho, estudo e vida pessoal;
2. Foque no essencial: nem tudo pode ser feito, escolha tarefas estratégicas e delegue ou elimine as menos críticas;
3. Conheça seu ritmo: identifique seus horários de maior energia para atividades complexas (estudo/trabalho) e respeite seus limites;
4. Use ferramentas: anote prazos, tempo gasto em tarefas e priorize com métodos como a matriz GUT (Grave, Urgente, Tendencioso);
5. Evite desperdícios: terceirize tarefas repetitivas para ganhar tempo produtivo;
6. Compromissos claros: assuma apenas o que pode cumprir e demandas alinhadas com seus objetivos;
7. Equilíbrio dinâmico: ajuste a atenção entre projetos, família e lazer, nem tudo precisa ser perfeito ao mesmo tempo;
8. Métricas pessoais: meça seu desempenho (exemplo: tempo para criar casos de teste) para melhorar estimativas futuras;
9. Autocuidado primeiro: você só ajuda outros se estiver bem física e mentalmente;
10. Compromisso com resultados: entregue valor ao cliente, equipe e empresa, mas sem sacrificar sua saúde ou integridade.

Resumo dos pontos-chave: autogerenciamento é sobre escolhas conscientes, foque no que traz impacto real e aprenda a otimizar recursos (tempo, energia, ferramentas).

- Comunicação verbal e não verbal

1. Impacto da comunicação:
 - Verbal (10%): palavras precisam ser claras e objetivas, especialmente em e-mails e documentações.
 - Tom de voz (35%): entonação e ênfase são cruciais para engajar o ouvinte. Evite monotonia ou excesso de volume.
 - Não verbal (55%): postura, gestos e expressões faciais transmitem mais que as palavras.
2. Adaptação ao contexto:
 - Formal (reuniões, e-mails) vs. informal (equipe próxima).
 - Ajuste a linguagem conforme o público (técnico, clientes, gestores).
 - Opte por fazer mapas mentais, isso evita problemas de interpretação e ajuda a guiar no momento da explicação.
3. Cuidados com comunicação escrita:
 - Evite CAPSLOCK (parece gritar) e cores agressivas (exemplo: vermelho).
 - Use imagens/diagramas para complementar textos complexos.
4. Primeiras impressões:
 - Aparência e ambiente (em entrevistas ou reuniões virtuais) influenciam a credibilidade.
 - Nos primeiros 100 dias em um novo trabalho, ações reforçam a imagem profissional.
5. Comunicação não verbal:
 - Gestos: fortalecem a mensagem, mas atenção a diferenças culturais (exemplo: "joinha" pode ofender).
 - Expressões faciais: alinhar com o discurso (exemplo: sorrir ao dar boas notícias).
 - Postura: braços cruzados = resistência; inclinar-se para frente = interesse.
 - Prefira fazer reuniões com até 8 pessoas, para que a maioria não fique como "ouvinte".
6. Para melhorar:
 - Treine interpretação de texto (leia resenhas, artigos).
 - Grave vídeos seu para analisar tom de voz e gestos.
 - Cuide do ambiente em que está.
 - Use pausas estratégicas para destacar pontos ou ouvir o outro.
 - Arrisque um pouco mais, conheça e se envolva com as pessoas de verdade.

Resumo dos pontos-chave: comunicação eficaz é consistência entre o que você diz, como diz e como seu corpo reforça a mensagem.

- Negociação

1. Negociação é essencial tanto na vida pessoal quanto profissional, impactando resultados a curto e longo prazo;
2. Abordagem "Ganha-Ganha" (Harvard): relações sustentáveis exigem que ambas as partes saiam satisfeitas. Se uma perde, ambas perdem indiretamente;
3. Riscos do "Ganha-Perde":
 - Fornecedores ou colegas explorados podem retaliar futuramente;
 - Ambiente tóxico: quem sempre "vence" discussões perde aliados quando precisar de ajuda;
4. Exemplo prático: negociar desconto com fornecedor pode levar a cobranças extras depois. Relações contínuas exigem equilíbrio;
5. "Perde-Perde": falha em acordos prejudica todos (exemplos: projetos não entregues, conflitos não resolvidos);
6. Como estruturar uma negociação:
 - Mapeie o que você precisa ganhar, o que pode ceder e seus limites;
 - Entenda as necessidades da outra parte;
7. Troca justa: só ceda algo se receber algo em retorno (exemplo: ajuda mútua entre colegas em sprints);

8. Sustentabilidade: parcerias duradouras valem mais que vitórias pontuais;

9. "Um bom negócio é bom para todos".

Resumo dos pontos-chave: negocie com empatia e visão estratégica, relações sólidas geram melhores resultados no longo prazo.

Produtividade

1. Comece gradualmente: como um atleta, inicie com pequenos hábitos (estudo/trabalho) e aumente progressivamente para evitar burnout;
2. Construa hábitos: produtividade é um "músculo" que se fortalece com prática e repetição diária;
3. Regra das 5 horas: dedique pelo menos 1h por dia (ou 5h/semana) para aprendizado contínuo;
4. Foque no presente: o hoje é o único momento sob seu controle, aja agora para melhorar seu futuro;
5. Exercite-se: atividade física melhora sono, memória e reduz estresse, impactando diretamente seu rendimento;
6. Rituais saudáveis: alimentação balanceada, hidratação e 7h de sono são bases para energia e clareza mental;
7. Evite pessoas negativas: se cerque por quem inspira e apoia seu crescimento, se afaste de críticos destrutivos;
8. Experimente técnicas: teste métodos de produtividade (exemplos: Pomodoro, GTD) e faça a adaptação para sua rotina;
9. Autoavaliação: reflita sobre como otimizar seu tempo, identifique e elimine desperdiçadores (exemplo: redes sociais);
10. Equilíbrio: produtividade não é sobre trabalhar mais, mas sobre priorizar o que traz resultados sustentáveis;
11. Erros como aprendizado: falhar faz parte do crescimento, quem não tenta não evolui. Analise seus erros para melhorar continuamente;
12. Metas claras: formalize seus objetivos (por escrito ou publicamente) para aumentar o comprometimento e receber apoio;
13. Gratidão: reconhecer o que você já conquistou reduz o estresse e melhora o foco para novos desafios;
14. Aproveite a jornada: a felicidade está no processo, não apenas no resultado final. Celebre pequenas vitórias;
15. Foco em soluções: enxergue problemas como oportunidades de crescimento. Quase tudo tem solução com criatividade e persistência;
16. Compartilhe seus sonhos: tornar objetivos públicos motiva e permite que outros contribuam com ideias e oportunidades;
17. Resiliência: situações difíceis trazem experiência, profissionais que superam adversidades se tornam mais completos;
18. Progresso, não perfeição: se compare apenas com sua versão anterior, buscando evolução constante.

Resumo dos pontos-chave: produtividade é consistência, pequenos ganhos diários levam a grandes transformações a longo prazo. Não é só técnica, é uma atitude de aprendizado contínuo e foco no que realmente importa.

• Fluxo contínuo

1. Ritmo sustentável: estabeleça uma capacidade realista de trabalho (exemplo: 15 pontos/sprint) para evitar sobrecarga e garantir qualidade;
2. Analogia do ônibus: correr pontualmente (como em deadlines) é aceitável, mas manter um sprint constante é insustentável, ajuste o ritmo para longo prazo;
3. Melhoria gradual: times evoluem como atletas, começam com metas alcançáveis, por exemplo, 15 pontos, e, com prática, aumentam para 20, 25 etc;
4. Paradoxo da velocidade: trabalhar mais rápido pode gerar retrabalho (como engarrafamentos). Reduzir a pressão aumenta a produtividade real.
5. Kanban: fazer mais com menos recurso
 - Visualização: Quadro com colunas (A Fazer, Em Construção, Pronto) para gerenciar tarefas;
 - Limites de WIP (Work in Progress): Restrinja tarefas em andamento para evitar gargalos (exemplo: máximo de 3 em construção);
 - Fluxo contínuo: priorize entregas incrementais (não espere o pacote final da sprint);
6. Exemplo real: times que entregam consistentemente 15 pontos/sprint podem, com tempo e confiança, atingir 20+ naturalmente;
7. Foco na capacidade atual: reconheça o estágio do time (como um corredor iniciante) e evite comparar com equipes de alta performance prematuramente;

8. Benefícios:

- Menos estresse e mais qualidade na entrega;
- Melhoria contínua sem burnout;
- Entregas mais frequentes e previsíveis.

Dica final: Fluxo contínuo é sobre constância – equilíbrio entre desafio e capacidade real, com evolução orgânica.

Resumo dos pontos-chave: fluxo contínuo é sobre constância, equilíbrio entre desafio e capacidade real, com evolução orgânica e organizada.

- Técnica pomodoro

1. Origem: criada por Francesco Cirillo nos anos 90, usa um timer (originalmente em formato de tomate - "pomodoro" em italiano) para gerenciar o tempo;
2. Estrutura básica:
 - 25 minutos de trabalho focado (1 pomodoro);
 - 5 minutos de pausa curta;
 - Após 4 pomodoros, uma pausa maior (15-30 minutos);
3. Objetivo: combater interrupções (notificações, colegas, distrações) e aumentar a produtividade;
4. Como aplicar:
 - Divida tarefas complexas em blocos de 25 minutos;
 - Use timers físicos ou apps (como Focus To-Do, Pomodoro Timer, Clockify, Be Focused);
 - Proteja os pomodoros: desative notificações e avise colegas sobre seu foco;
5. Benefícios:
 - Melhora a concentração e reduz a procrastinação;
 - Transforma "tempos mortos" (esperas, deslocamentos) em oportunidades produtivas;
 - Facilita o progresso contínuo em tarefas longas (exemplo: 16h de trabalho = 32 pomodoros).
6. Desafios:
 - Resistir a interrupções urgentes (chefes, emergências);
 - Manter a disciplina nas pausas (evitar estendê-las);
7. Dica chave: 15 pomodoros/semana (cerca de 1h30/dia) já geram impacto significativo em 3-6 meses.

Exemplo prático: Em vez de adiar uma tarefa de 2 dias, divida-a em 32 pomodoros e execute em blocos ao longo da semana.