Пирамида тестирования

помогает понять как выстраивать тесты на проекте

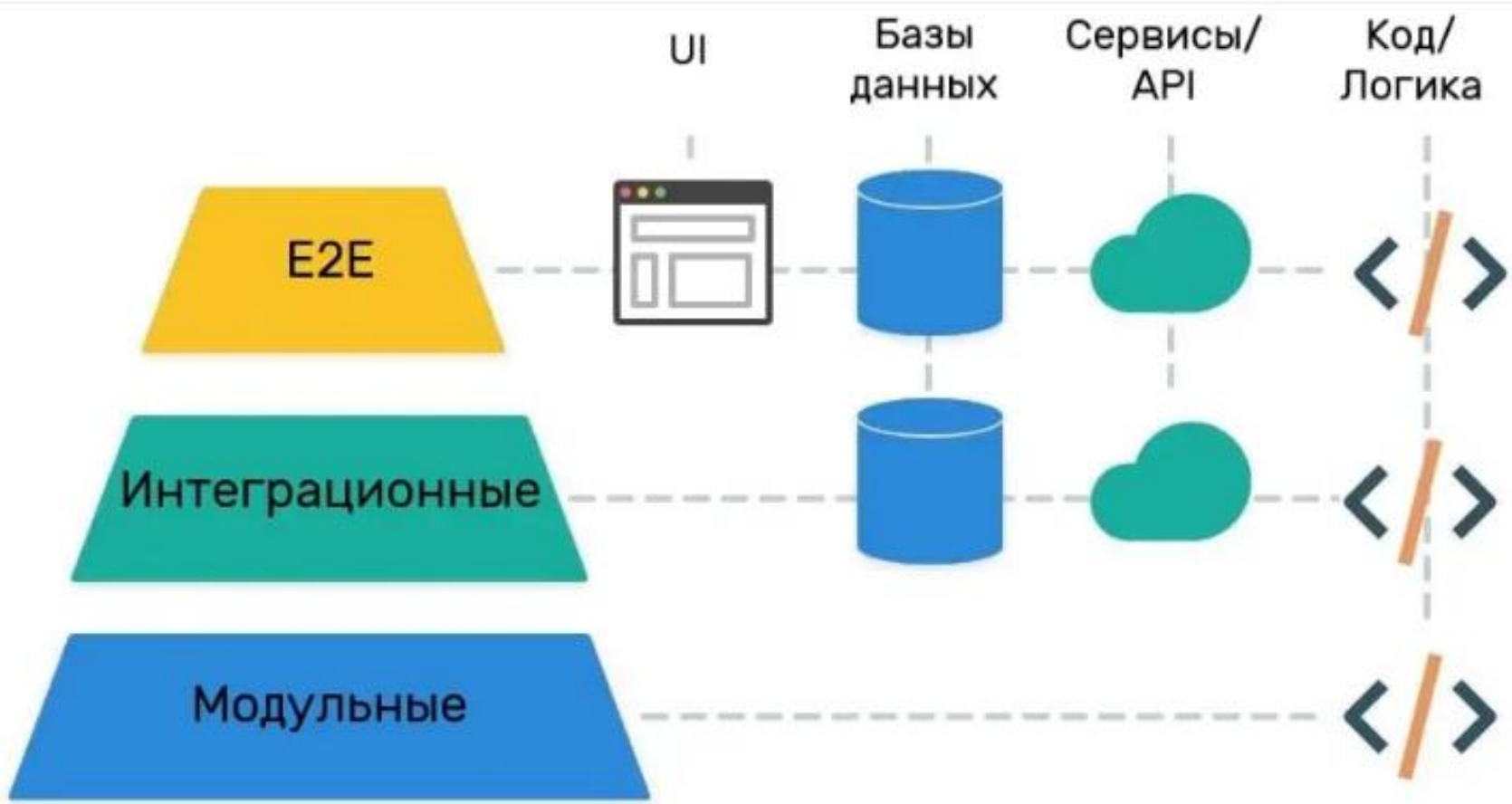
Стандартная тестовая пирамида

Тесты верхнего уровня (E2E), проверяют соблюдение требований к интерфейсу и общих требований к продукту. Из-за сложности и трудоемкости их количество по возможности стараются ограничить

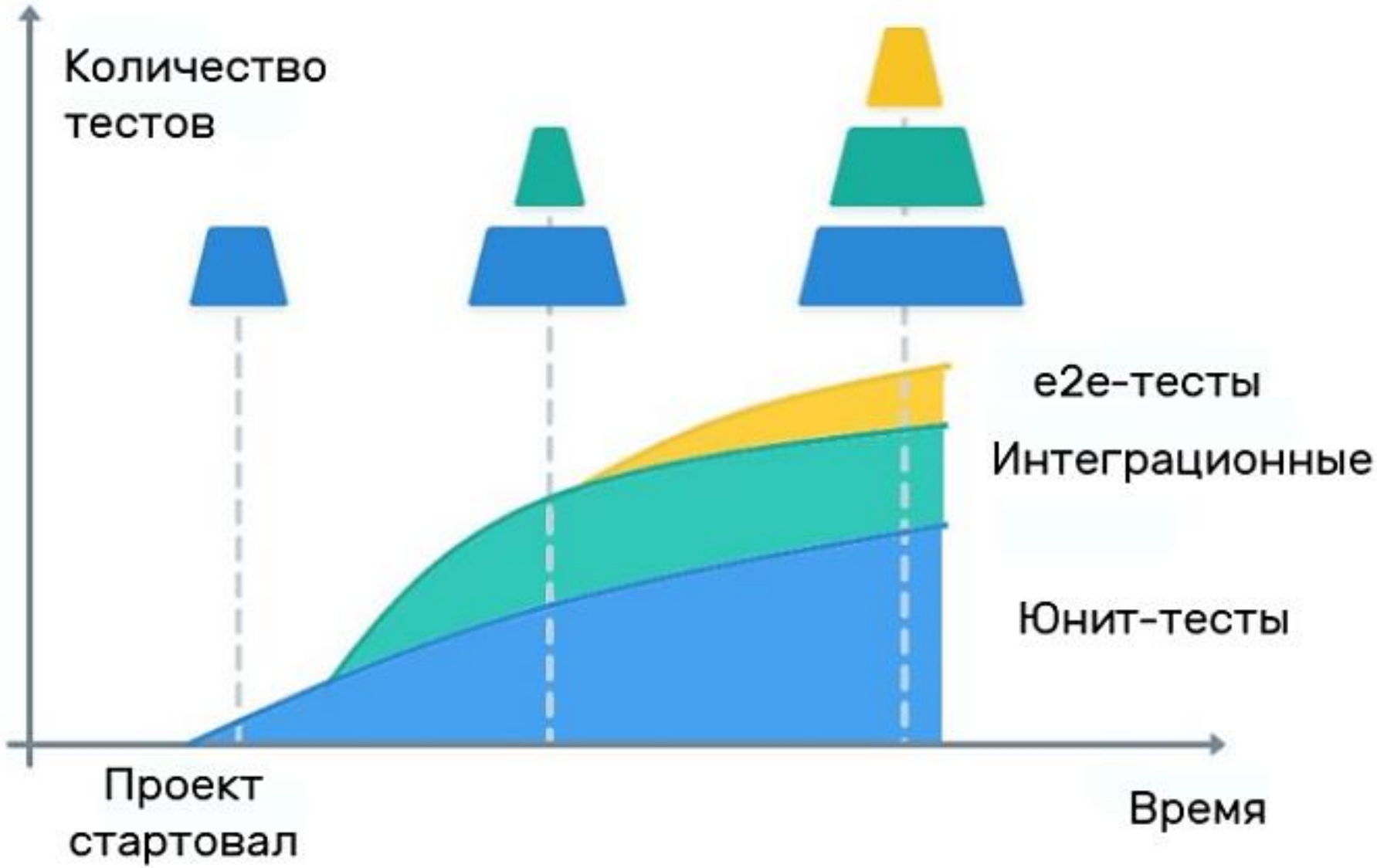
Имеются в виду интеграционные тесты, проверяющие взаимодействия между юнитами и более крупными чем юнит сущностями — компонентами.

Юнит-тестов больше чем всех остальных [вместе взятых], потому что в приложении как правило много модулей, соответственно модульных тестов требуется много, и они простые.

Самая старая, и все еще самая широко применяемая, как бы эталонная тестовая стратегия, впервые подробно описанная в книге Майкла Кона “*Succeeding with Agile*”.

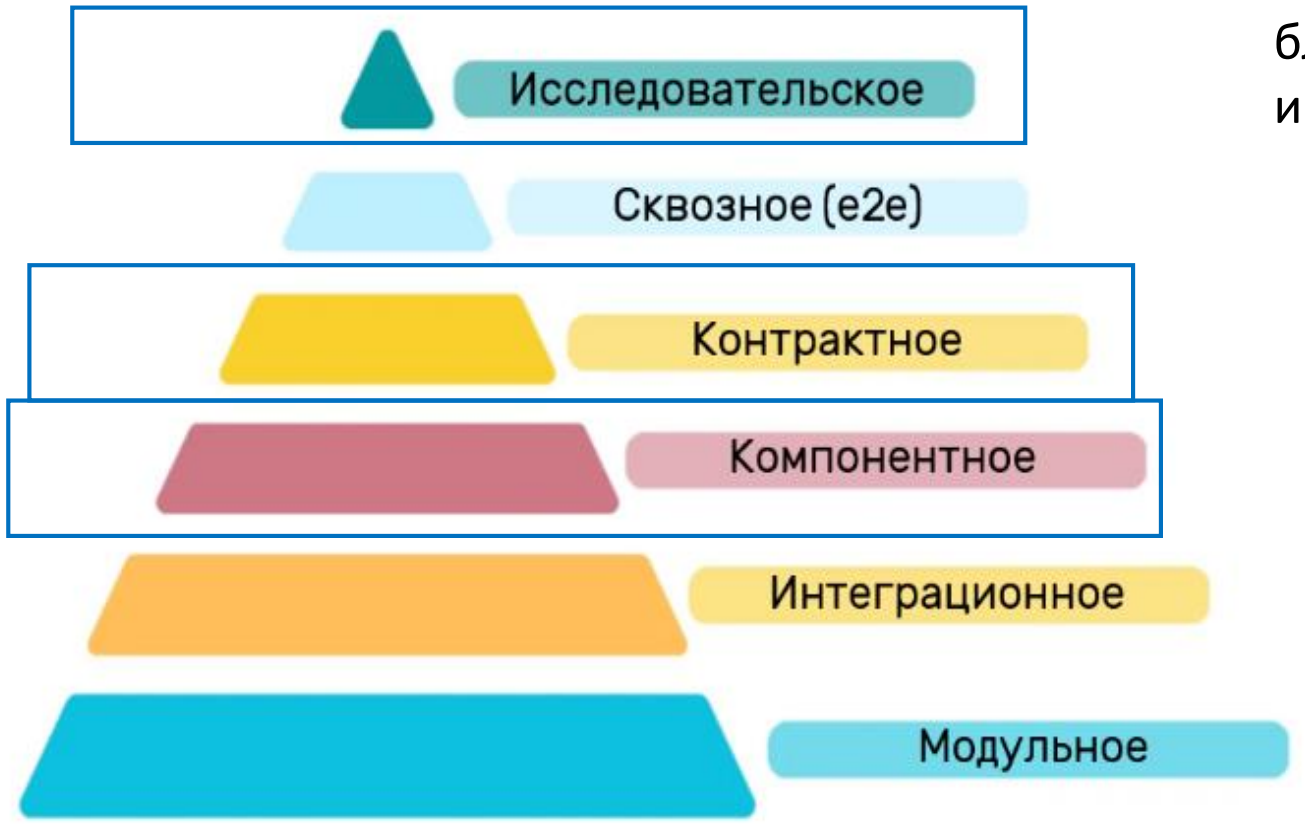


Тип тестов	Скорость выполнения
Юнит-тесты	0,01-0,1 секунды
Сервисные (интеграционные) тесты	1 секунда
Сквозные (e2e) и UI-тесты	10 секунд


Количество
тестов
e2e-тесты
Интеграционные
Юнит-тесты
Проект
стартовал
Время

Расширенная пирамида тестирования

Со временем стандартную пирамиду начали расширять добавлением отдельных **интеграционных контрактных тестов**, и **компонентных тестов**. Также в расширенной пирамиде появляются **ручные исследовательские тесты**.



При этом сохраняется закономерность «чем выше, тем меньше тестов»; чем ближе к концу цикла, тем сложнее тесты и меньше их количество.

Контрактное тестирование - это методология проверки, что две подсистемы (чаще всего два микросервиса) совместимы и могут беспроблемно коммуницировать. контракта» соблюдают обязательства.

Песочные часы (DevOps)

В DevOps делают упор на тщательный мониторинг процессов. DevOps рассчитан на разработку с частыми деплоями. Поэтому к стандартной тестовой пирамиде «присоединяются» последующие процессы **Logging-Monitoring-Alerting**.

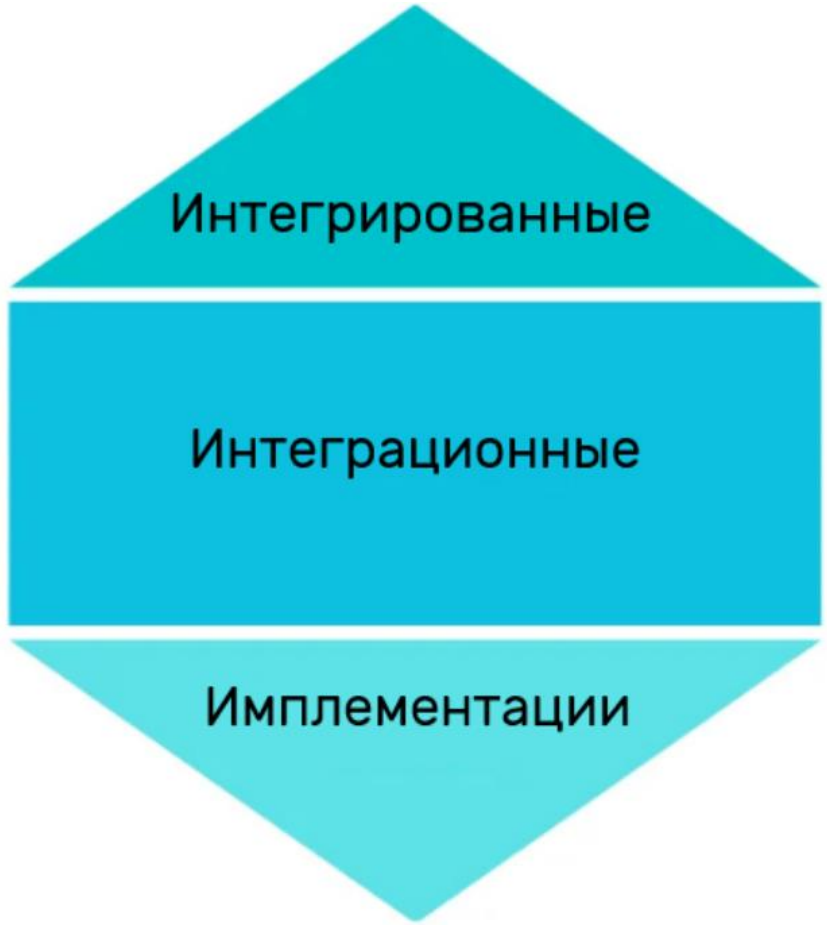
- **Журналирование (Logging)** — налаженное протоколирование (фиксация) информации о происходящем в приложении
- **Мониторинг (Monitoring)** — автоматизированное наблюдение за состоянием приложения и его серверной части
- **Оповещения (Alerting)** — налаженный процесс автоматических оповещений о проблемных точках приложения, чтобы их по возможности немедленно устранить

Сота

Специфическую тестовую пирамиду представили в компании Spotify в 2018г. Такая стратегия заточена под микросервисную архитектуру, в которой самая большая сложность часто состоит не как в самом сервисе, а как в его взаимодействиях с другими сервисами.

Сота, состоит из 3 слоев:

- **Интегрированные** (integrated) тесты — тесты, которые *проходят* или *падают* в зависимости от правильности взаимодействий с другой (внешней) системой.
- **Интеграционные** (integration) тесты проверяют правильность работы основного сервиса в более изолированном виде чем предыдущие, фокусируясь на точках взаимодействия и совершенствуя их работу.
- **Тесты имплементации** сфокусированы на изолированных частях кода, как бы специфический аналог юнит-тестирования.

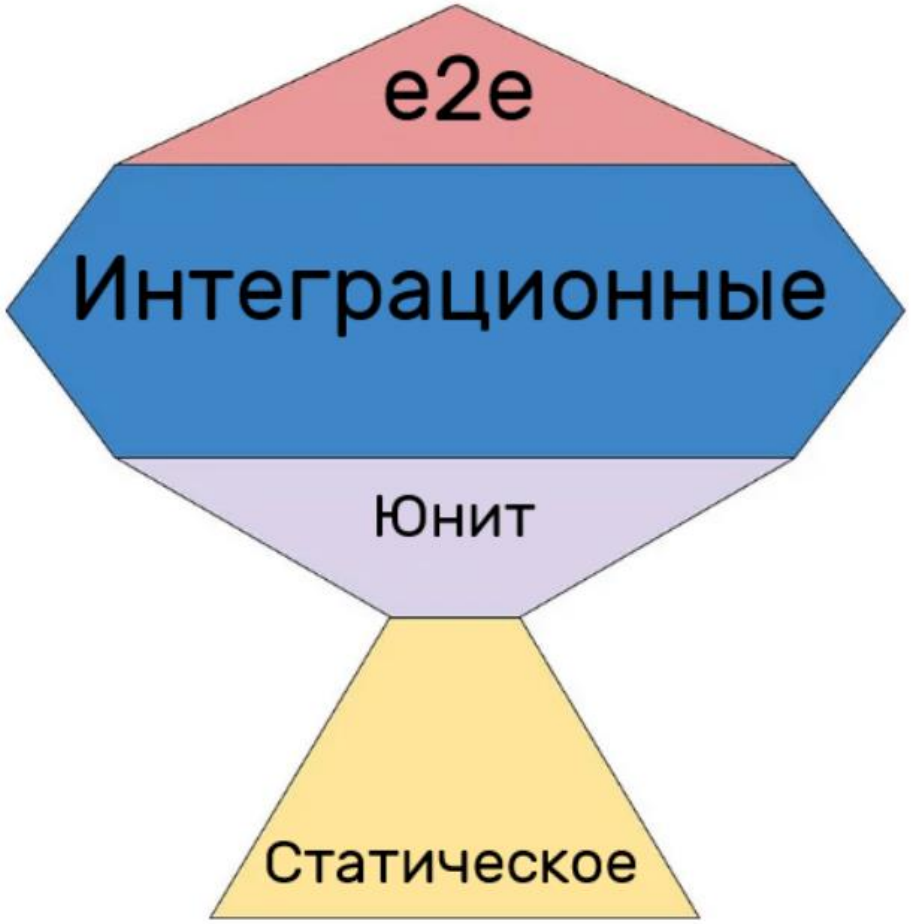


Кубок

Концепция представлена в 2018 году, признанным экспертом в QA Кеннетом Доддсом.

Автор концепции, исходя из своего опыта, полагает, что по мере продвижения проекта к верхушке тестовой пирамиды, уверенность в качестве тестов (а значит и количество их) должна быть максимальной на среднем уровне пирамиды — на ее интеграционном уровне.

Но также большое внимание должно уделяться предварительному уровню пирамиды — статическому тестированию, которое проводится перед модульным.



Вулкан

Еще один вариант тестовой пирамиды — «извергающийся вулкан». Это часто встречающийся вариант уровней тестирования в современной разработке.

Количество исследовательских и ручных тестов может быть достаточно большим, а вообще неопределенно и непредсказуемо, как и длительность ручного тестирования.

Поклонники стратегии «вулкана» полагают, что классическая пирамида уровней тестирования уже отжила свое, плохо учитывает рыночные риски проектов, что нужно больше опираться на риск-тестирование, и что старое-доброе ручное тестирование (включая исследовательское), и в отдаленном будущем нельзя заменить никогда и ничем.

Перевернутая пирамида тестирования

Часто встречается на практике.

Даже сейчас, с повсеместным внедрением Agile/Scrum/DevOps и прочих полезных методологий, разделение труда между Dev- и QA-отделами все еще очень отчетливое, **тестировщики не всегда имеют доступ к кодовой базе, и если разработчиков не заставляют писать юнит-тесты, то «фундаментные тесты»**, *будь то юнит-, компонентные или интеграционные*, **просто некому писать**, и все возможное с QA продукта делают тестировщики. Вследствие чего количество «тестов верхнего уровня» закономерно увеличивается.

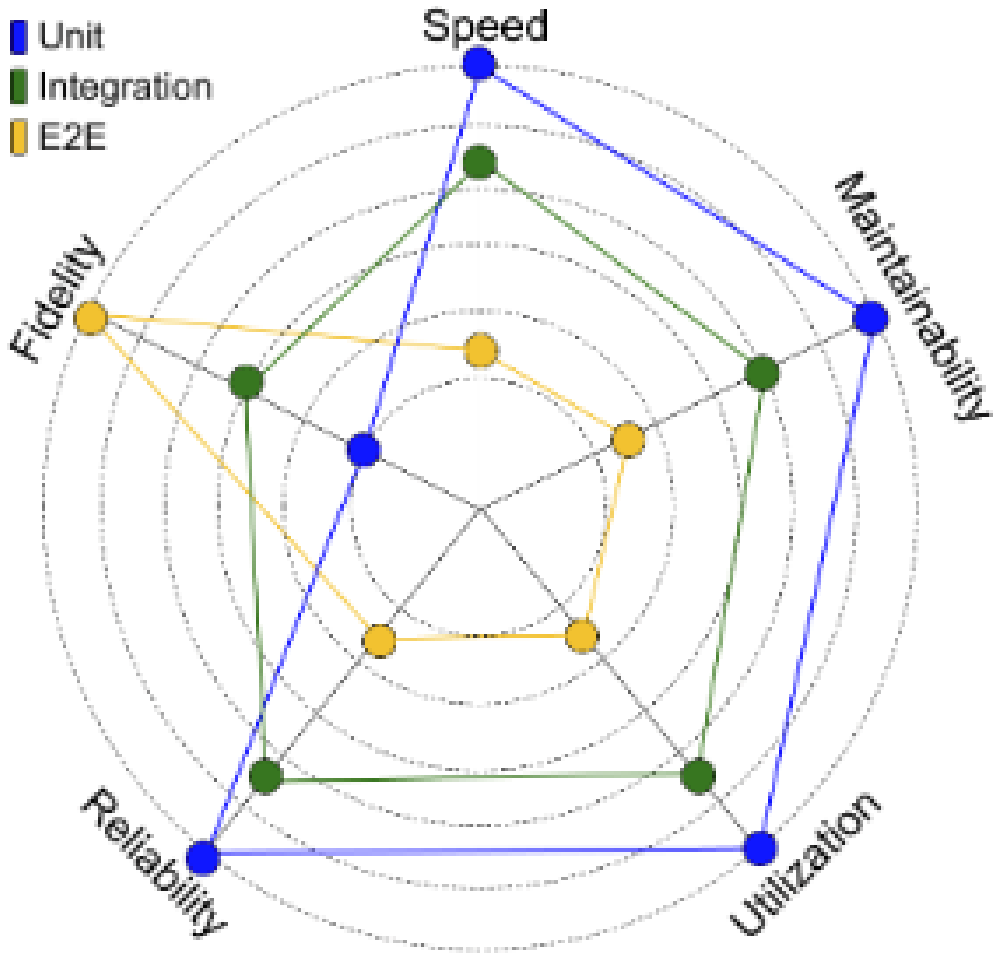
Также в современных проектах заказчики склонны требовать «релиз к сроку любой ценой», не особо вникая в перипетии взаимоотношений между отделами. Команды вынуждены регулярно демонстрировать заказчикам «то что есть», и быстрые e2e-тесты от QA-отдела — единственный разумный выход.

Мнемоника SMURF — простой способ запомнить 5 характеристик, которые необходимо учитывать при балансировке растущего тестового набора:

Скорость (**Speed**): Юнит-тесты намного быстрее других и могут выполняться чаще.

Удобство обслуживания (**Maintainability**): Совокупные затраты на отладку и поддержку тестов (всех типов) быстро растут. В большой системе больше кода, а это значит, что система больше подвержена изменению зависимостей и дрейфу требований, что приводит к увеличению объема работ по обслуживанию.

Утилизация ресурсов (**Utilization**): Тесты, использующие меньше ресурсов (память, диск, процессор), обходятся дешевле. Хороший тестовый набор оптимизирует использование ресурсов таким образом, чтобы оно не росло сверхлинейно с увеличением количества тестов. Юнит-тесты обычно обладают отличными характеристиками утилизации, потому что в них используются тестовые дублеры, и проверяются маленькие части системы.



https://testengineer.ru/smurf-google-test-pyramid/

Надежность (**Reliability**): Надежные тесты падают только при обнаружении реальной проблемы. Разбор нестабильных тестов в поисках проблем требует времени и ресурсов на повторное выполнение тестов. По мере роста системы и количества тестов в ней растет неопределенность (следовательно и нестабильность тестов), и ваш набор теряет надежность.

Точность (**Fidelity**): Точные тесты соответствуют реальным условиям работы системы (например, реальной нагрузке на БД и реальному трафику) и лучше предсказывают поведение на продакшене. Интеграционные и сквозные тесты лучше отражают реальные условия, в то время как юнит-тесты вынуждены моделировать свое окружение, что приводит к расхождению между ожиданиями от тестов и реальностью.