Персонализация возможностей C++ IDE с помощью функциональных модулей clangd

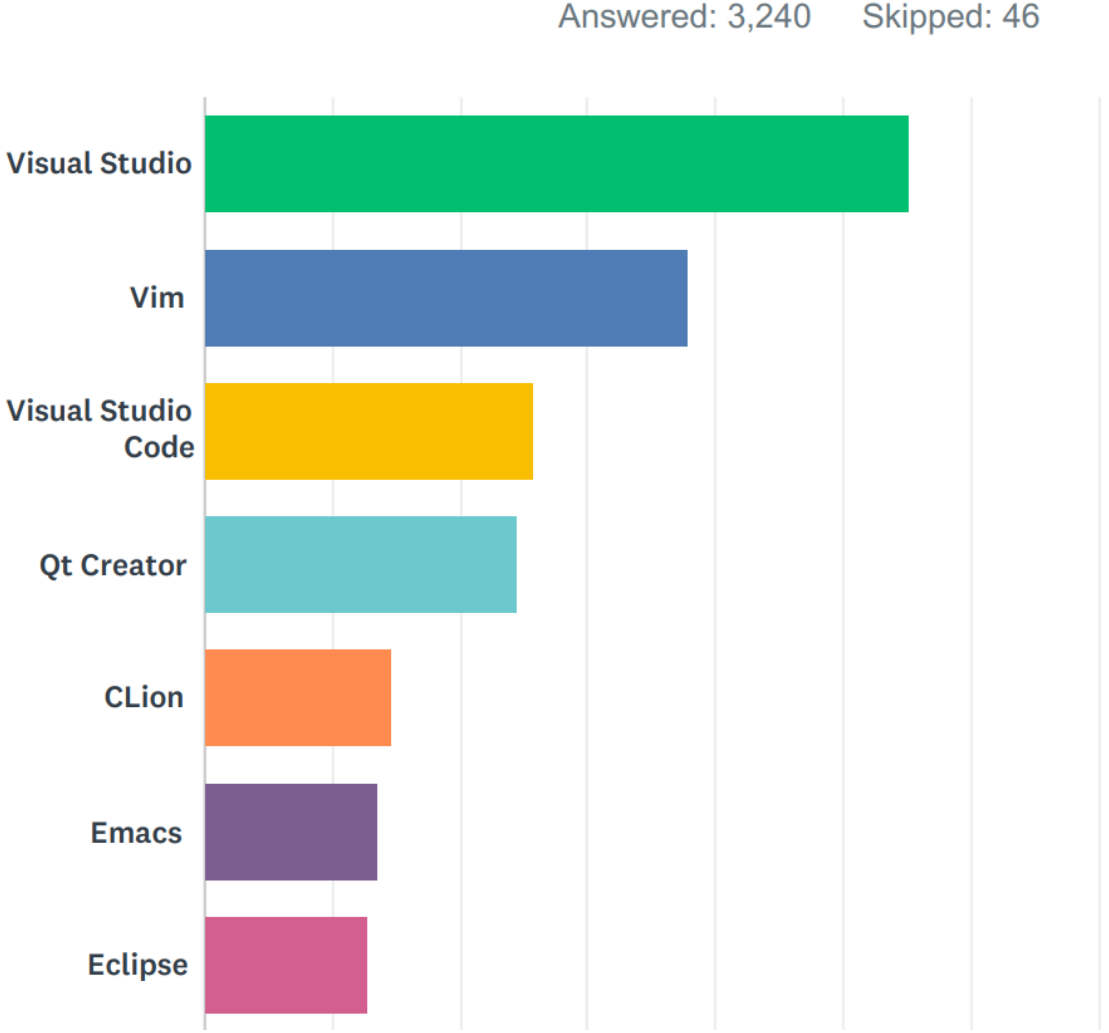
АЛЕКСАНДР ПЛАТОНОВ

План доклада

- Существует ли подходящая всем IDE?
- Что такое clangd?
- Функциональные модули и их возможности
- Примеры реализаций
- Выводы

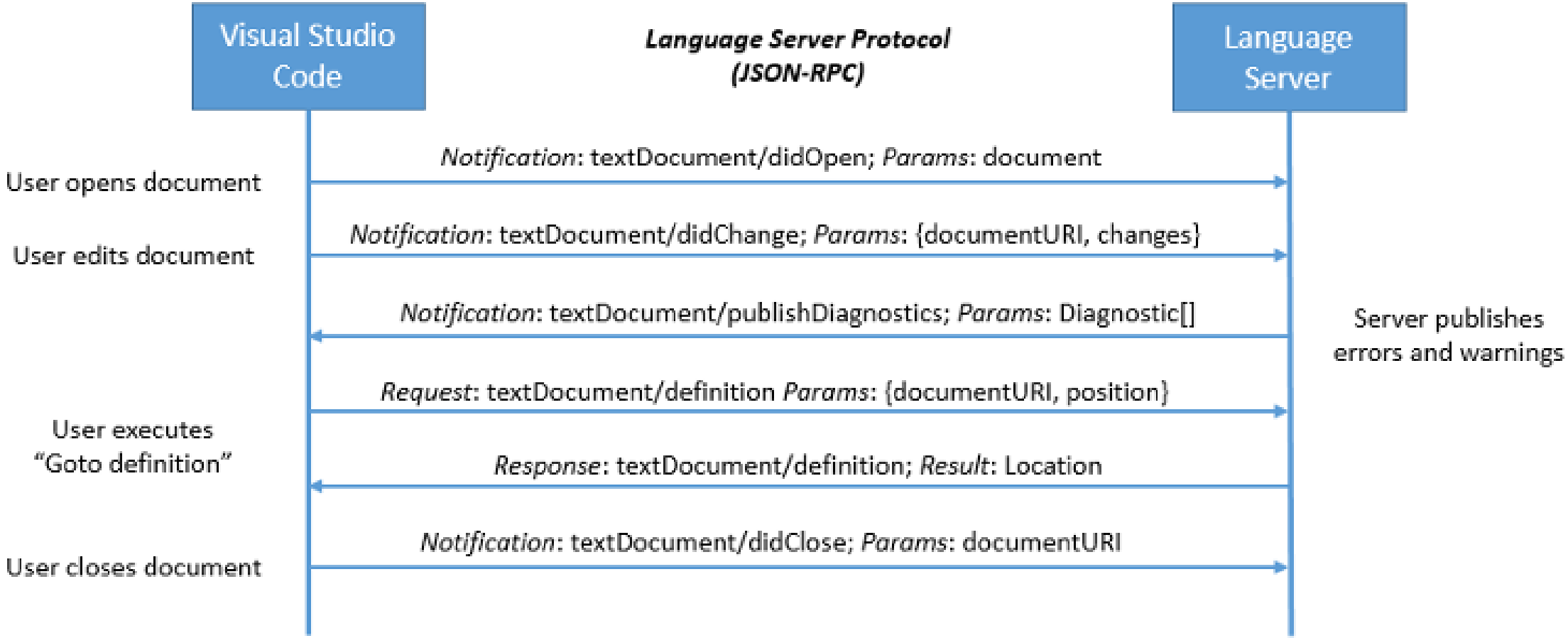
“Идеальная” C++ IDE

IDE/Editor	Responses
Visual Studio	55.28%
Vim	37.93%
Visual Studio Code	25.77%
Qt Creator	24.41%
CLion	14.66%
Emacs	13.55%
Eclipse	12.78%
Sublime	12.78%
XCode	12.78%
Other	12.22%



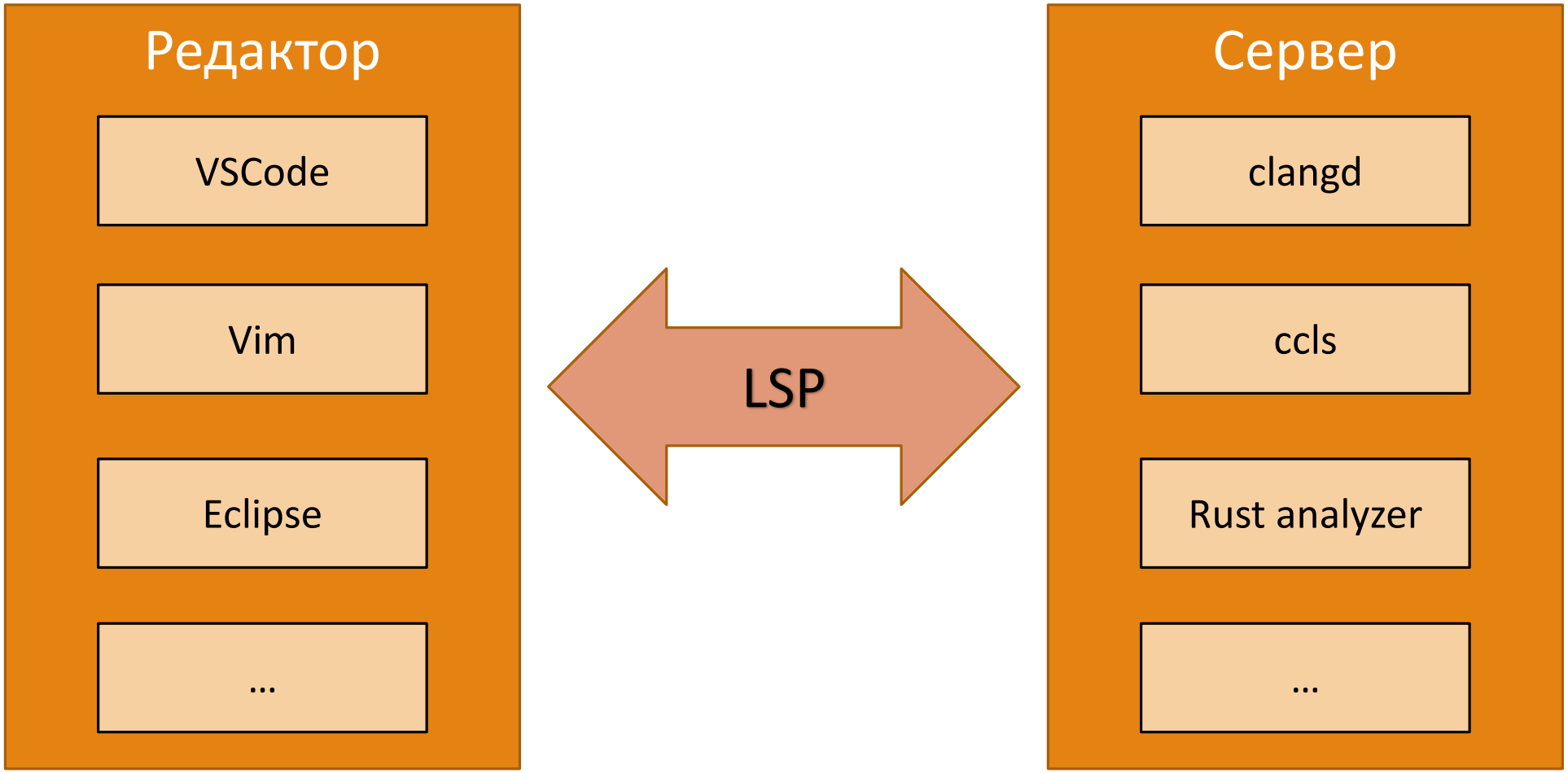
https://isocpp.org/files/papers/CppDevSurvey-2018-02-summary.pdf

Language server protocol



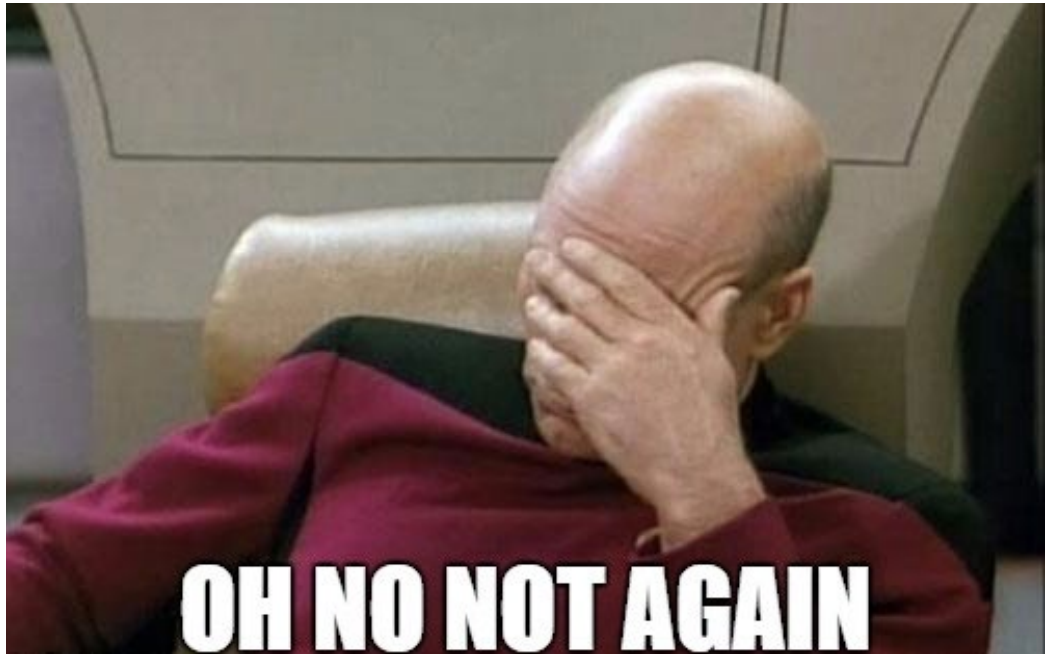
https://microsoft.github.io/language-server-protocol/

Language server protocol



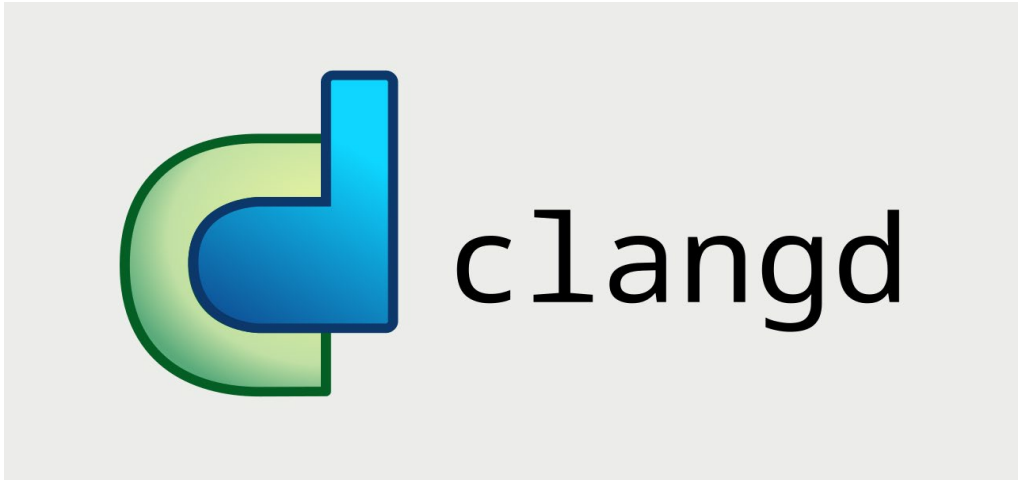
“Идеальный” языковой сервер

- Привычки разработчиков
- Разные задачи – разные приоритеты по функционалу
- Встроенные статические проверки
- Разное ожидаемое поведение



clangd – “teach your editor C++”

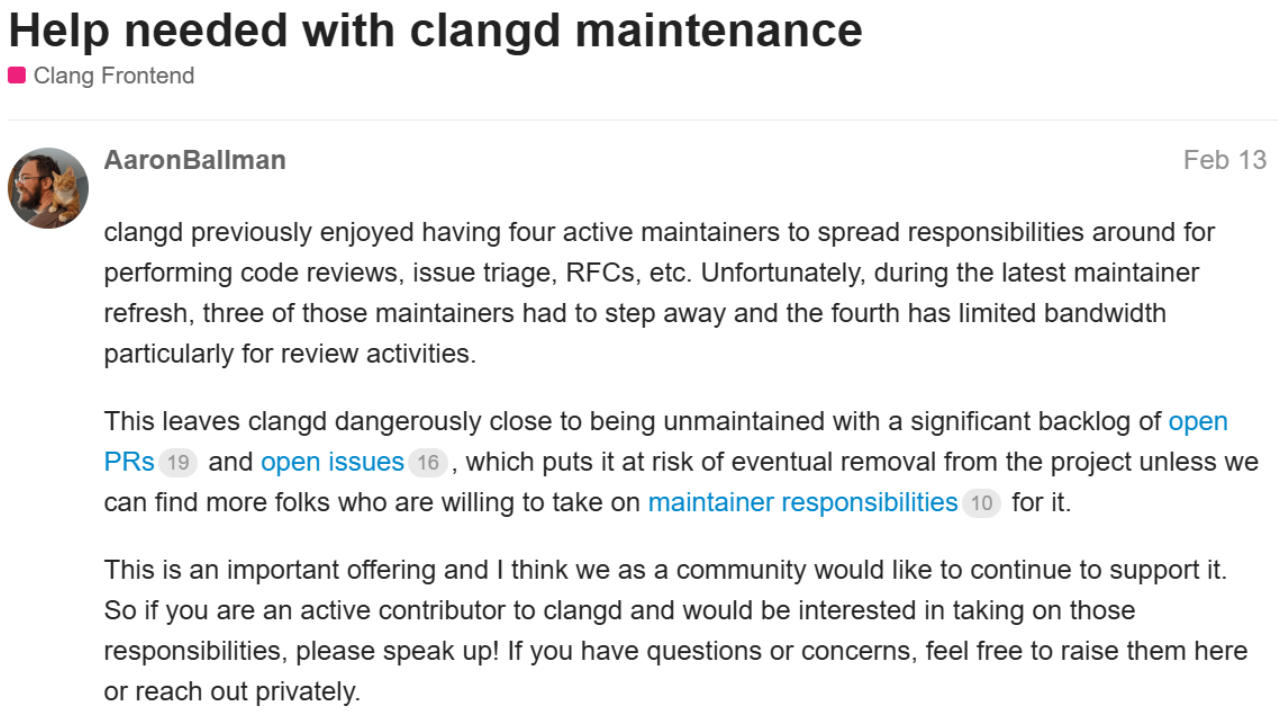
- Языковой сервер для C/C++/Objective C
  - Работает с разными редакторами через плагин
    - ✓Language Server protocol
- Предоставляет функционал IDE
  - Дополнение кода
  - Поиск определения/декларации
  - Семантическая раскраска
  - Рефакторинг
  - Форматирование
  - …



https://clangd.llvm.org/

Изменение в clangd: PR на github

- Не все изменения хочется открывать
- Не все изменения могут быть приняты
- Долгое ревью



Help needed with clangd maintenance

Clang Frontend

AaronBallman Feb 13

clangd previously enjoyed having four active maintainers to spread responsibilities around for performing code reviews, issue triage, RFCs, etc. Unfortunately, during the latest maintainer refresh, three of those maintainers had to step away and the fourth has limited bandwidth particularly for review activities.

This leaves clangd dangerously close to being unmaintained with a significant backlog of open PRs 19 and open issues 16, which puts it at risk of eventual removal from the project unless we can find more folks who are willing to take on maintainer responsibilities 10 for it.

This is an important offering and I think we as a community would like to continue to support it. So if you are an active contributor to clangd and would be interested in taking on those responsibilities, please speak up! If you have questions or concerns, feel free to raise them here or reach out privately.

https://discourse.llvm.org/t/help-needed-with-clangd-maintenance/89820

Изменение в clangd: свой форк

- Разрешение конфликтов при синхронизации
- Запутанная история коммитов

Функциональные модули clangd

- Нет официальной документации
  - ✓Doxygen документация
  - ✓Исходный код
- Нет реальных примеров в открытых источниках
  - ✓Есть тесты
  - ✓Реальные функциональные модули существуют!

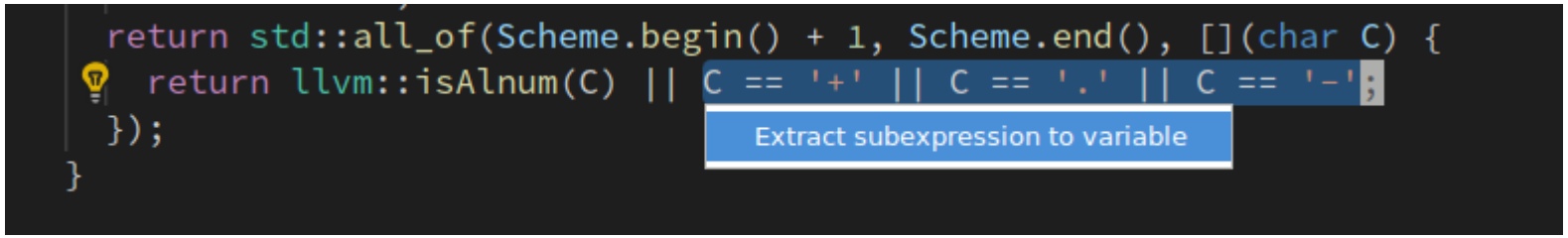
https://github.com/llvm/llvm-project/blob/main/clang-tools-extra/clangd/FeatureModule.h

Возможности

- Уровень Language Server Protocol
  - Расширение
  - Переопределение существующих обработчиков!
- Обход синтаксического дерева
  - Собственные статические проверки
  - Сбор данных для других тулов
- Обработка диагностик
  - Сбор информации
  - Корректировка!
- Добавление методов рефакторинга (tweakов)
  - А так же удаление!

clangd Tweaks

- Небольшие встроенные плагины для рефакторинга
- Контекстно-зависимы
- Работают через интерфейс Code Actions (textDocument/codeAction)

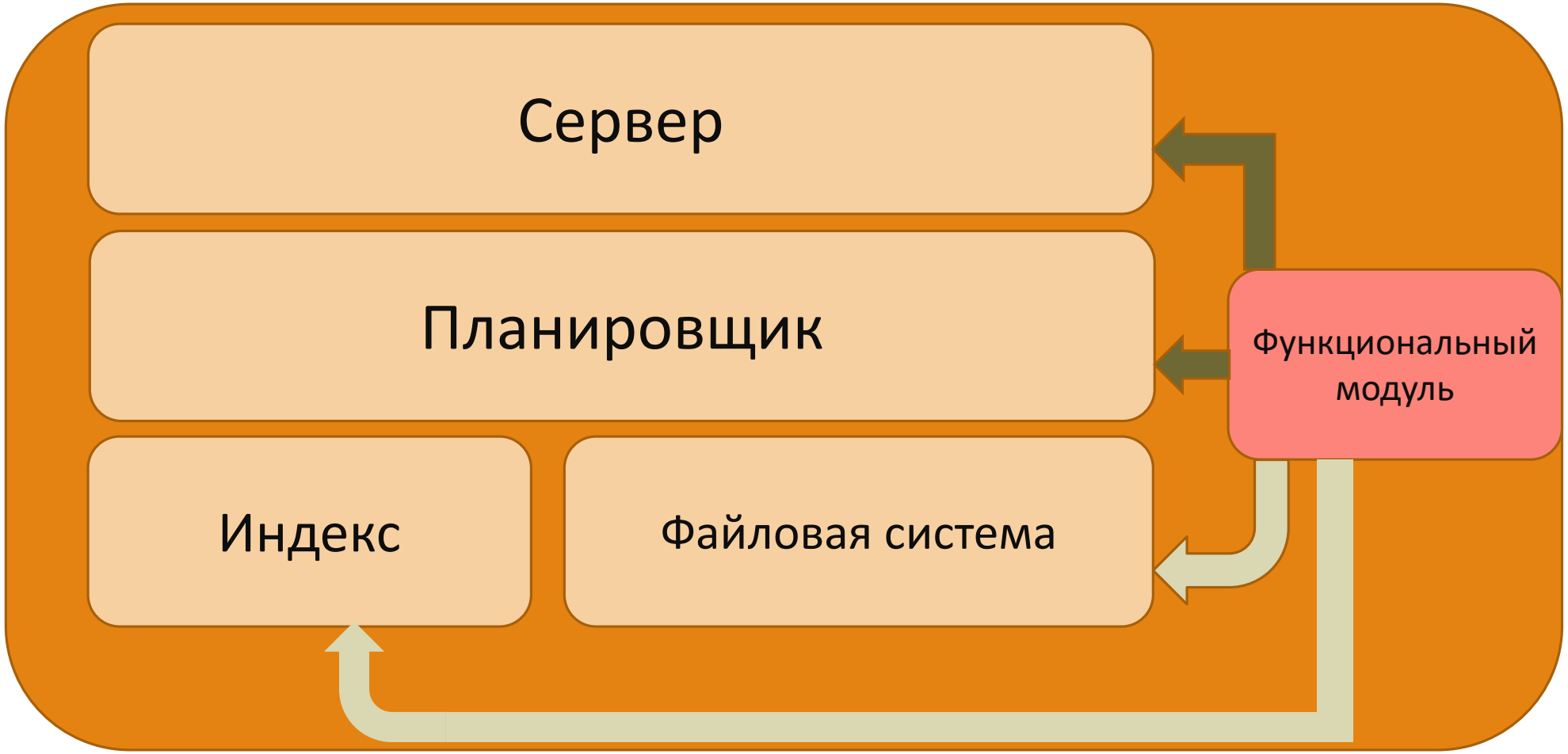


https://clangd.llvm.org/design/code#code-actions

https://microsoft.github.io/language-server-protocol/specifications/lsp/3.17/specification/#textDocument_codeAction

Архитектура clangd

Архитектура clangd



Класс функционального модуля

```
class FeatureModule {
…
  virtual void initializeLSP(LSPBinder &Bind,  const llvm::json::Object &ClientCaps,  llvm::json::Object &ServerCaps) {}
  virtual void contributeTweaks(std::vector<std::unique_ptr<Tweak>> &Out) {}
  struct ASTListener {
     virtual void beforeExecute(CompilerInstance &CI) {}
     virtual void sawDiagnostic(const clang::Diagnostic &, clangd::Diag &) {}
  };
  virtual std::unique_ptr<ASTListener> astListeners() { return nullptr; }
…
};
```

https://github.com/llvm/llvm-project/blob/main/clang-tools-extra/clangd/FeatureModule.h

Класс функционального модуля

```
class FeatureModule {
…
  struct Facilities {
    TUScheduler &Scheduler;
    const SymbolIndex *Index;
    const ThreadsafeFS &FS;
  };
  /// Called by the server to prepare this module for use
  void initialize(const Facilities &F); // ClangdLSPServer::initialize() => Mod.initialize()
…
};
```

https://github.com/llvm/llvm-project/blob/main/clang-tools-extra/clangd/FeatureModule.h

Хуки

- Приём запроса инициализации по LSP

  LSP метод ‘initialize’

  ClangdLSPServer::onInitialize() => Mod.initializeLSP()

- Получение списка tweak’ов

  LSP метод ‘textDocument/codeAction’

  ClangdLSPServer::onCodeAction() => getAllTweaks() => Mod.contributeTweaks()

- Построение AST

  Разные LSP методы, например ‘textDocument/hover’

  ClangdLSPServer::* => TUScheduler::runWithAST() => Mod.astListeners()

Добавление функционального модуля

```
class TestFeatureModule final : public FeatureModule {

....

....

};

static FeatureModuleRegistry::Add<TestFeatureModule> X("test-feature",
                                                       "Test feature module");
```

https://discourse.llvm.org/t/rfc-registry-for-feature-modules/87733

https://github.com/llvm/llvm-project/pull/154836

Добавление функционального модуля

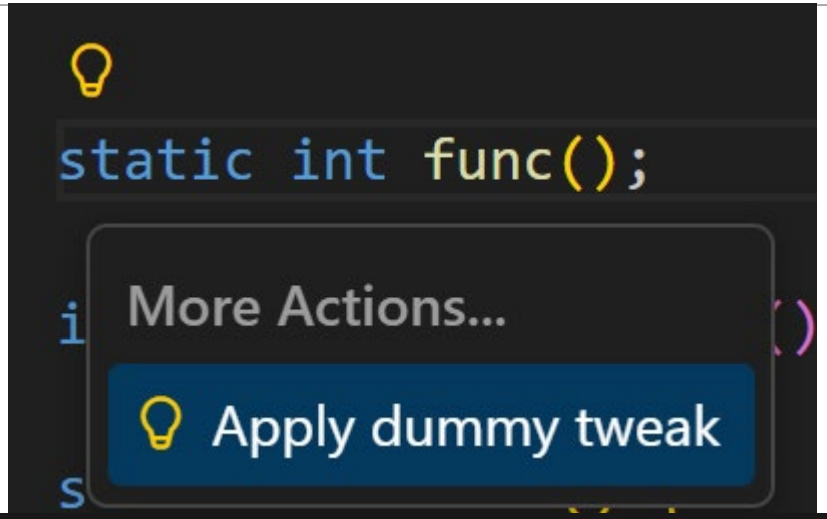
```
int clangdMain(int argc, char *argv[]) {
...
  FeatureModuleSet ModuleSet = FeatureModuleSet::fromRegistry();
   if (ModuleSet.begin() != ModuleSet.end())
     Opts.FeatureModules = &ModuleSet;
...
  ClangdLSPServer LSPServer(*TransportLayer, TFS, Opts);
...
}
```

Добавление tweak’а

```
class TestFeatureModule final : public FeatureModule {
  static constexpr const char *TweakID = "DummyTweak";
  struct DummyTweak final : public Tweak {
    const char *id() const override { return TweakID; }
    bool prepare(const Selection &) override { return true; }
    Expected<Effect> apply(const Selection &) override {
      return error("not implemented");
    }
    std::string title() const override { return "Apply dummy tweak"; }
    llvm::StringLiteral kind() const override {
      return CodeAction::REFACTOR_KIND;
    }
  };

public:
  void contributeTweaks(std::vector<std::unique_ptr<Tweak>> &Out) override {
    Out.emplace_back(new DummyTweak);
  }
};
```

Добавление tweak’а




```
<-- workspace/executeCommand(35)
ASTWorker running ApplyTweak on version 1 of /home/pam/llvm-project/build/test.

--> reply:workspace/executeCommand(35) 2 ms, error: not implemented
>>> {"error":{"code":-32001,"message":"not implemented"},"id":35,"jsonrpc":"2.
```

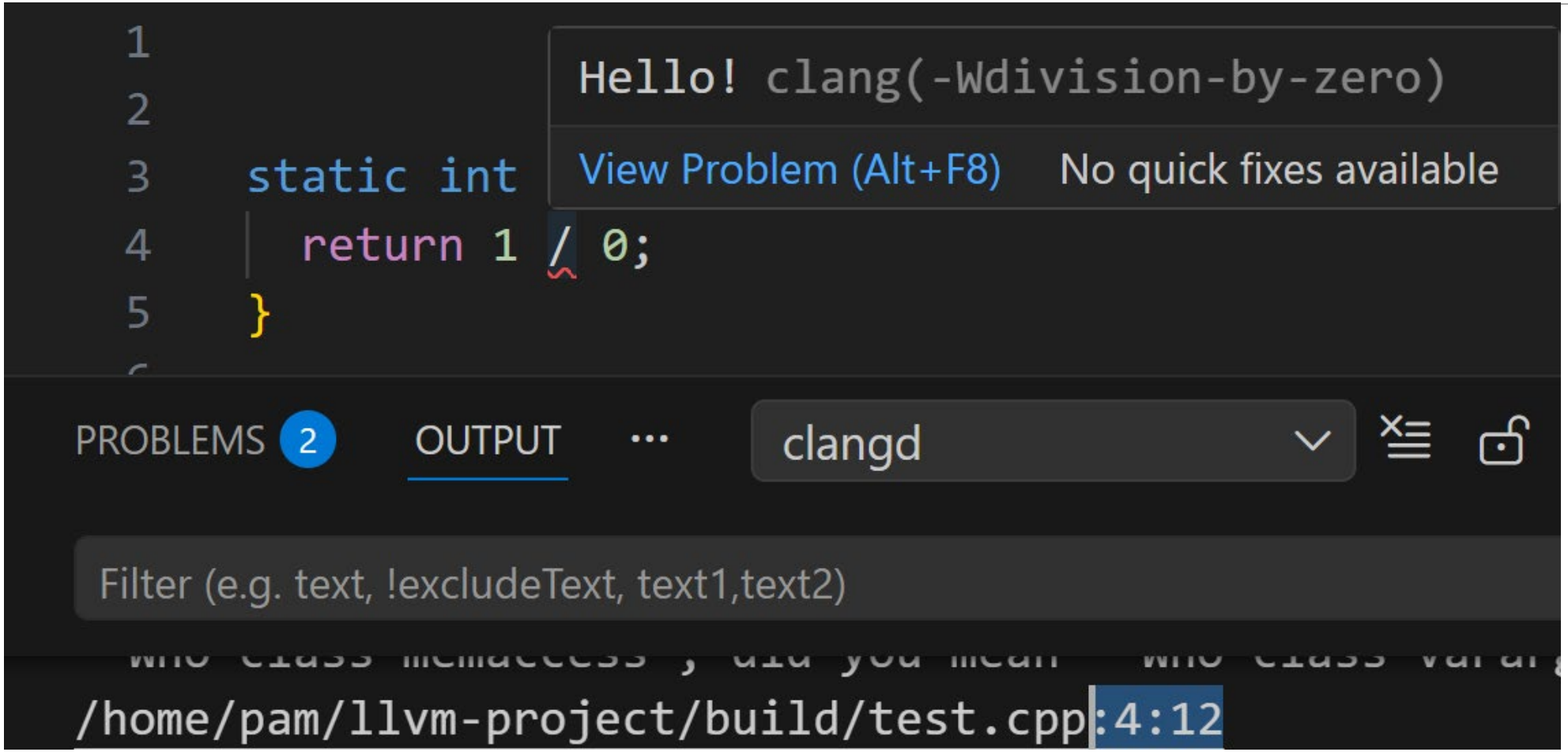
An unknown error occurred. Please consult the log for more details.

Обработка диагностик

```
class TestFeatureModule final : public FeatureModule {
  struct Listener : public FeatureModule::ASTListener {
    void sawDiagnostic(const clang::Diagnostic &Info,
                       clangd::Diag &Diag) override {
      Info.getLocation().dump(Info.getSourceManager());
      Diag.Severity = DiagnosticsEngine::Error;
      Diag.Message = "Hello!";
    }
  };

public:
  std::unique_ptr<ASTListener> astListeners() override {
    return std::make_unique<Listener>();
  }
};
```

Обработка диагностик



Обход синтаксического дерева

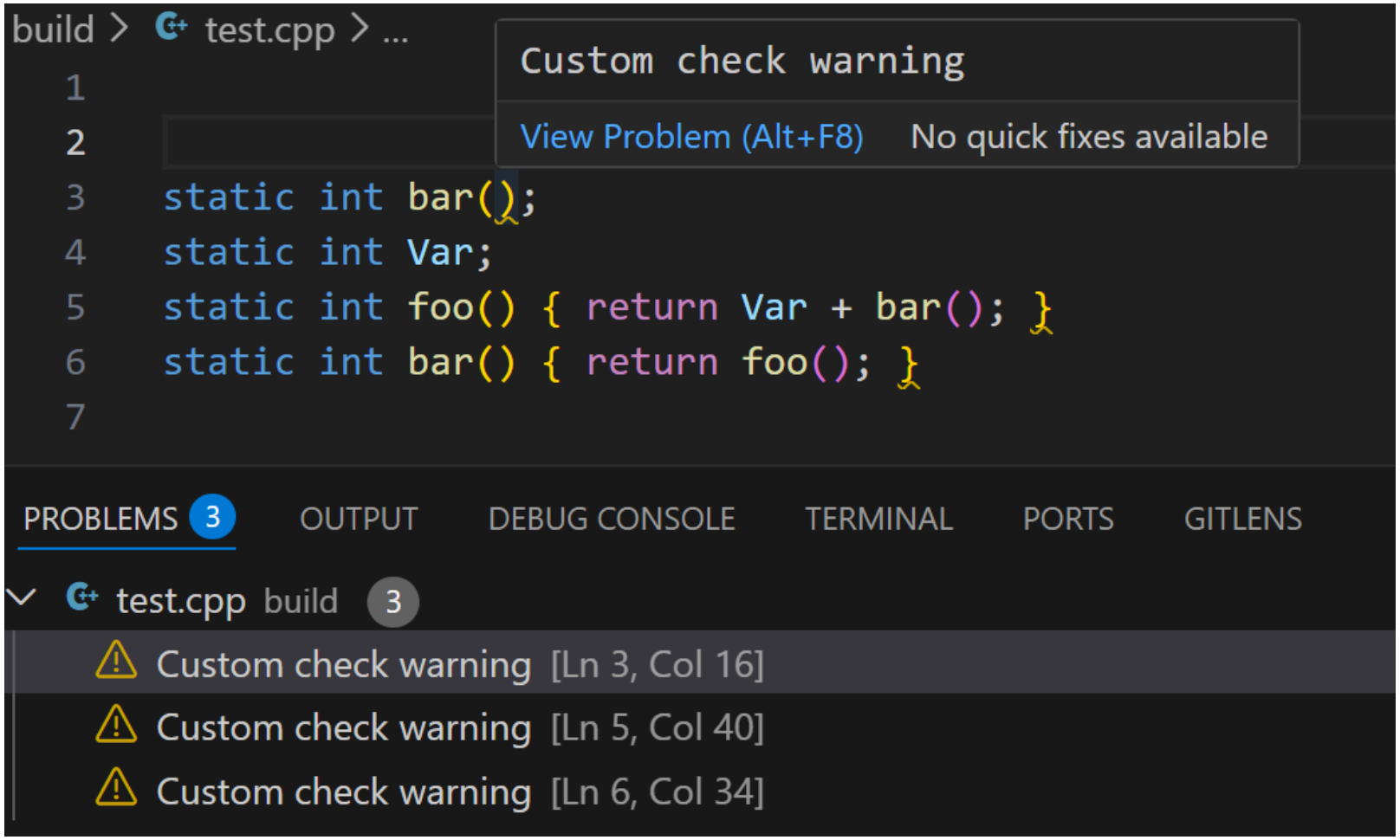
```
class TestFeatureModule final : public FeatureModule {
  struct Listener : public FeatureModule::ASTListener {
    class CustomCheck : public ASTConsumer {
      DiagnosticsEngine &DiagEngine;
    public:
      CustomCheck(DiagnosticsEngine &DiagEngine) : DiagEngine(DiagEngine) {}
      bool HandleTopLevelDecl(DeclGroupRef DG) override {
        for (auto *D : DG)
          if (auto *FD = dyn_cast<FunctionDecl>(D)) {
            DiagEngine.Report(
                FD->getSourceRange().getBegin(),
                DiagEngine.getDiagnosticIDs()->getCustomDiagID(
                    DiagnosticIDs::Warning, "Custom check warning"));
          }
        return true;
      }
    };
```

Обход синтаксического дерева

```
    void beforeExecute(CompilerInstance &CI) override {
      std::vector<std::unique_ptr<ASTConsumer>> Consumers;
      Consumers.push_back(CI.takeASTConsumer());
      Consumers.push_back(std::make_unique<CustomCheck>(CI.getDiagnostics()));
      CI.setASTConsumer(std::make_unique<MultiplexConsumer>(std::move(Consumers)));
    }
  };

public:
  std::unique_ptr<ASTListener> astListeners() override {…
  }
};
```

Обход синтаксического дерева



Расширение протокола

```
class TestFeatureModule final : public FeatureModule {
  void onHover(const TextDocumentPositionParams &Params,
               Callback<std::optional<Hover>> Reply) {
    findHover(Params.textDocument.uri.file(), Params.position,
              [Reply = std::move(Reply)](
                  llvm::Expected<std::optional<HoverInfo>> H) mutable {
                if (!H) return Reply(H.takeError());
                if (!*H) return Reply(std::nullopt);
                Hover R;
                R.contents.kind = MarkupKind::Markdown;
                R.range = (*H)->SymRange;
                R.contents.value = "Hello!\n" + (*H)->present(R.contents.kind);
                return Reply(std::move(R));
              });
  }
```

https://microsoft.github.io/language-server-protocol/specifications/lsp/3.17/specification/#textDocument_hover

Расширение протокола

```
public:
  void initializeLSP(LSPBinder &Bind, const llvm::json::Object &ClientCaps,
                     llvm::json::Object &ServerCaps) override {
    Bind.method("textDocument/hover", this, &TestFeatureModule::onHover);
  }
};
```

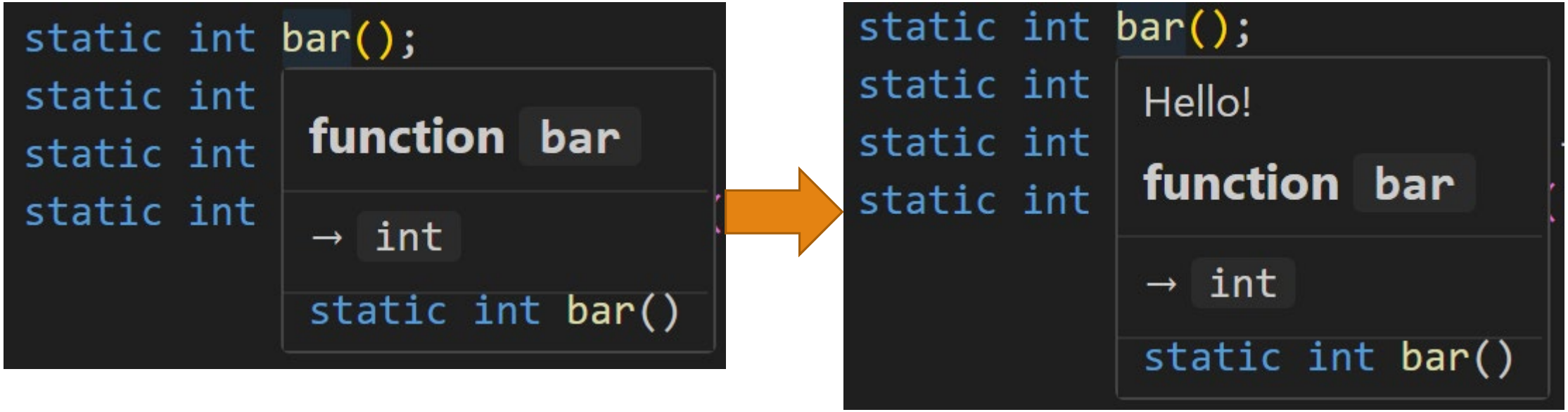
https://microsoft.github.io/language-server-protocol/specifications/lsp/3.17/specification/#textDocument_hover

Расширение протокола

```
class TestFeatureModule final : public FeatureModule {
  void findHover(PathRef File, Position Pos,
                 Callback<std::optional<HoverInfo>> CB) {
    auto Action = [File = File.str(), Pos, CB = std::move(CB),
                   this](llvm::Expected<InputsAndAST> InpAST) mutable {
      if (!InpAST)
        return CB(InpAST.takeError());
      format::FormatStyle Style = getFormatStyleForFile(
          File, InpAST->Inputs.Contents, *InpAST->Inputs.TFS, false);
      CB(clangd::getHover(InpAST->AST, Pos, std::move(Style),
                          facilities().Index));
    };

    facilities().Scheduler.runWithAST("Hover", File, std::move(Action),
                                      TUScheduler::NoInvalidation);
  }
  void onHover(const TextDocumentPositionParams &Params,
               Callback<std::optional<Hover>> Reply) {
```

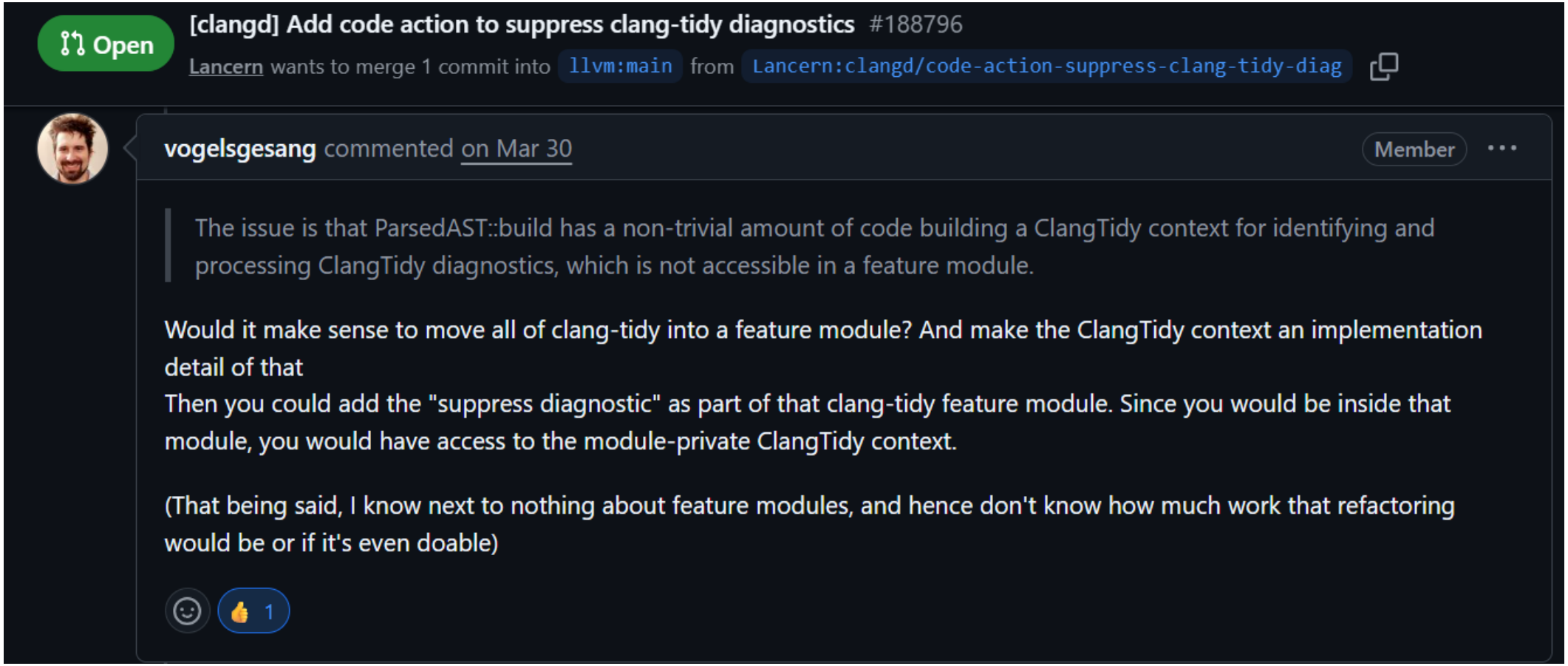
Расширение протокола



Расширение API функциональных модулей

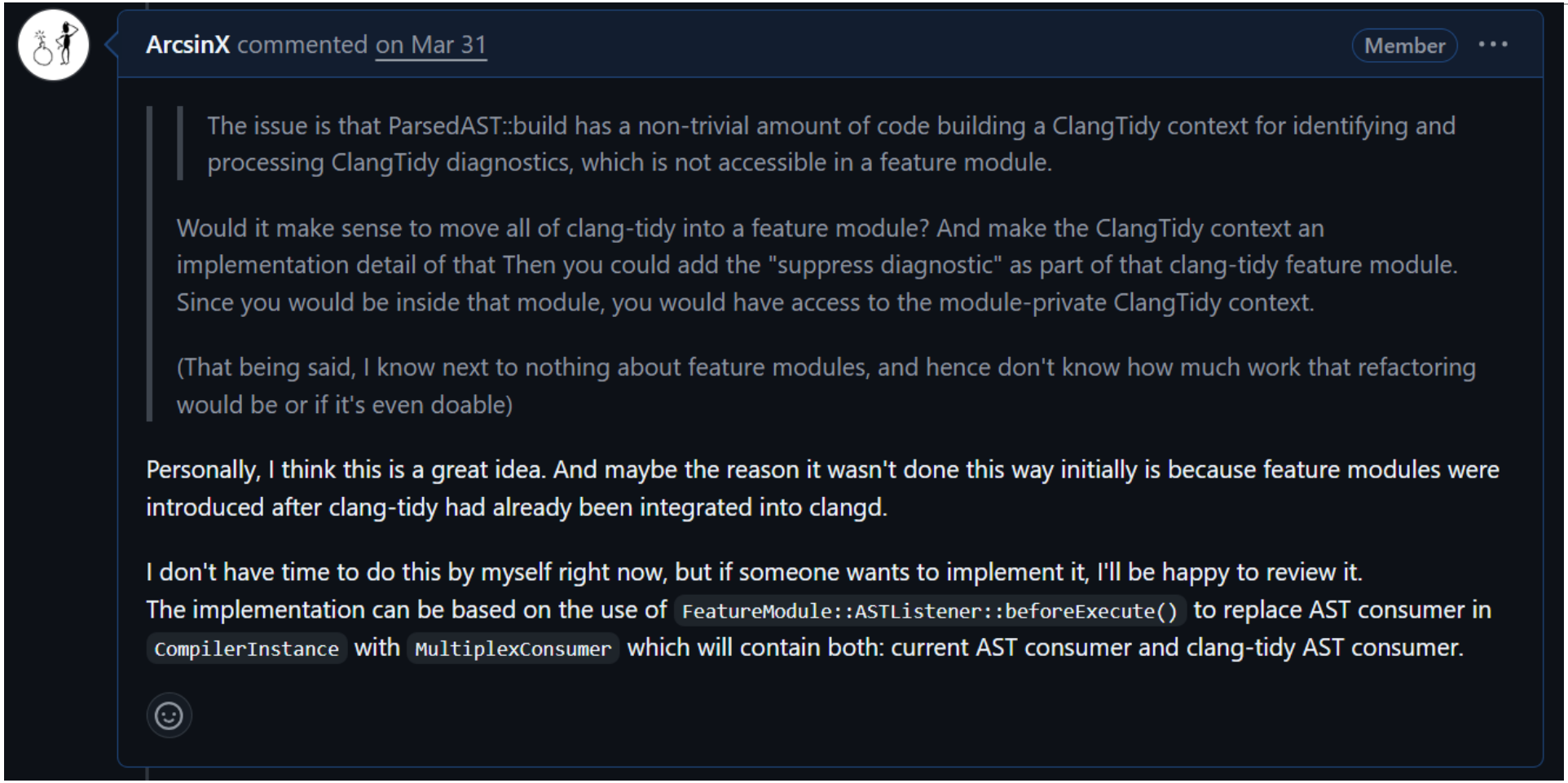
- Добавление хуков
  - Вызвать метод X для всех модулей при событии Y
    - ✓Добавить метод X
    - ✓Найти место для события Y
- Предоставление данных
  - Добавление в Facilities
    - ✓Например, указатель на базу данных компиляции

clangd, clang-tidy и функциональные модули



https://github.com/llvm/llvm-project/pull/188796

clangd, clang-tidy и функциональные модули



ArcsinX commented on Mar 31 — Member

> The issue is that ParsedAST::build has a non-trivial amount of code building a ClangTidy context for identifying and processing ClangTidy diagnostics, which is not accessible in a feature module.
>
> Would it make sense to move all of clang-tidy into a feature module? And make the ClangTidy context an implementation detail of that Then you could add the "suppress diagnostic" as part of that clang-tidy feature module. Since you would be inside that module, you would have access to the module-private ClangTidy context.
>
> (That being said, I know next to nothing about feature modules, and hence don't know how much work that refactoring would be or if it's even doable)

Personally, I think this is a great idea. And maybe the reason it wasn't done this way initially is because feature modules were introduced after clang-tidy had already been integrated into clangd.

I don't have time to do this by myself right now, but if someone wants to implement it, I'll be happy to review it.
The implementation can be based on the use of `FeatureModule::ASTListener::beforeExecute()` to replace AST consumer in `CompilerInstance` with `MultiplexConsumer` which will contain both: current AST consumer and clang-tidy AST consumer.

https://github.com/llvm/llvm-project/pull/188796

Ограничения и сложности

- Только статическая линковка
  - Относительно легко добавить динамическую
  - https://discourse.llvm.org/t/rfc-registry-for-feature-modules/87733
- Нет доступа к диагностикам/синтаксическому дерево на этапе индексирования
  - Можно исправить, но уже сложнее
- Нестабильность C++ clang API

Модульный языковой сервер как общее решение

- Базовая версия
  - с функционалом, подходящим ~~почти~~ всем
  - относительно небольшой размер
  - меньше потребление памяти
  - лучше производительность
- Дополнительные функциональные модули
  - Набор для проекта
  - Индивидуальные

Спасибо!

Вопросы?

Александр Платонов
email: arcsinx45@gmail.com
tg: @arcsinx45