presentation sponsored by KDAB

from the free world of KDE

NATURAL TASK SCHEDULING

Using Futures and Continuations

Qt Developer Days 2013

Ivan Čukić

ivan.cukic@kde.org
ivan.fomentgroup.org/blog

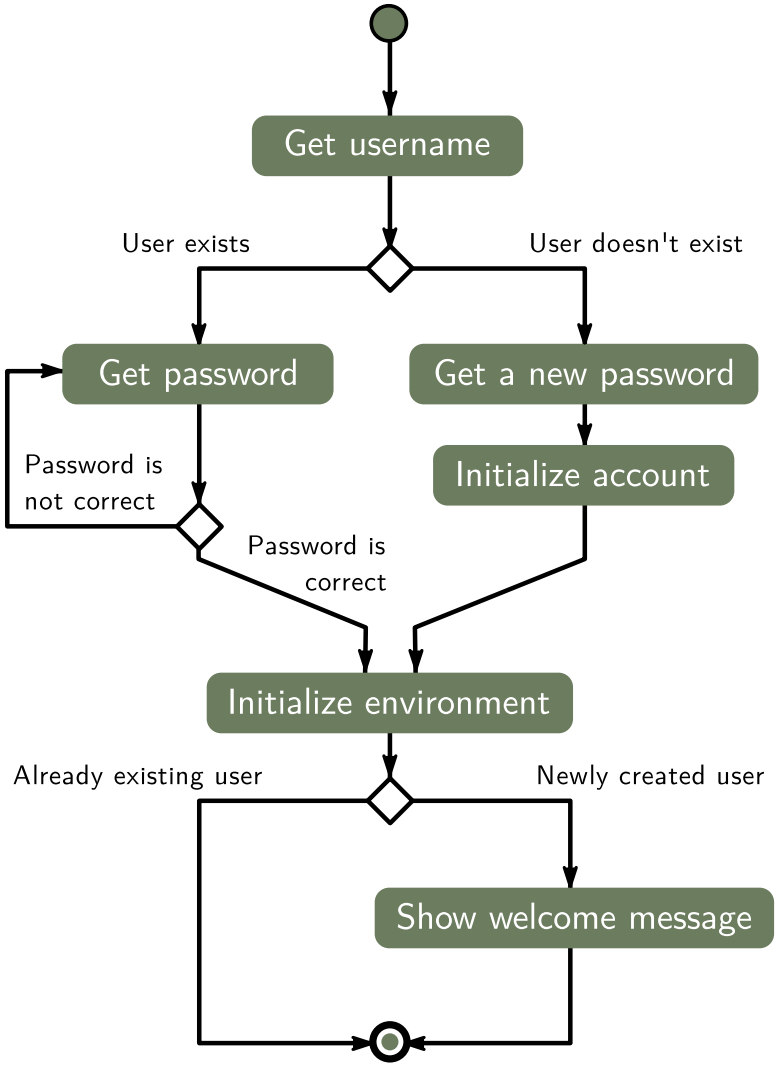
THE PROBLEM

Meet Jeff

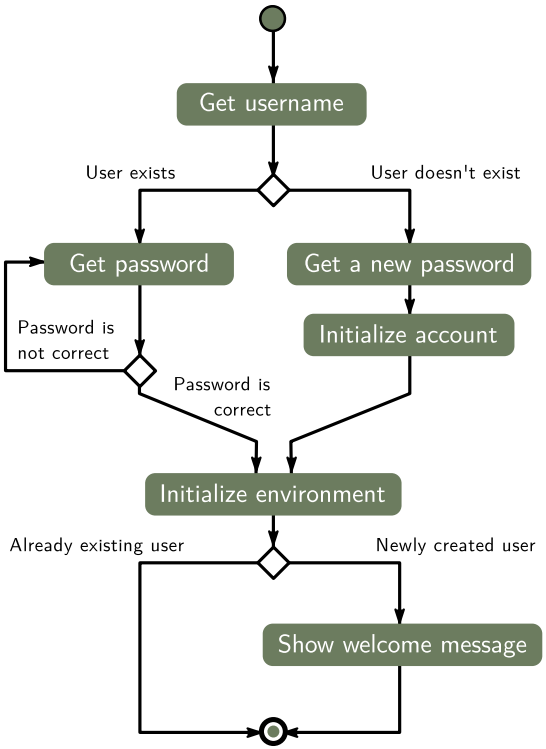
Out of Control

Reasons for Waiting

MEET JEFF


Get username
User exists
User doesn't exist
Get password
Get a new password
Password is not correct
Initialize account
Password is correct
Initialize environment
Already existing user
Newly created user
Show welcome message

MEET JEFF

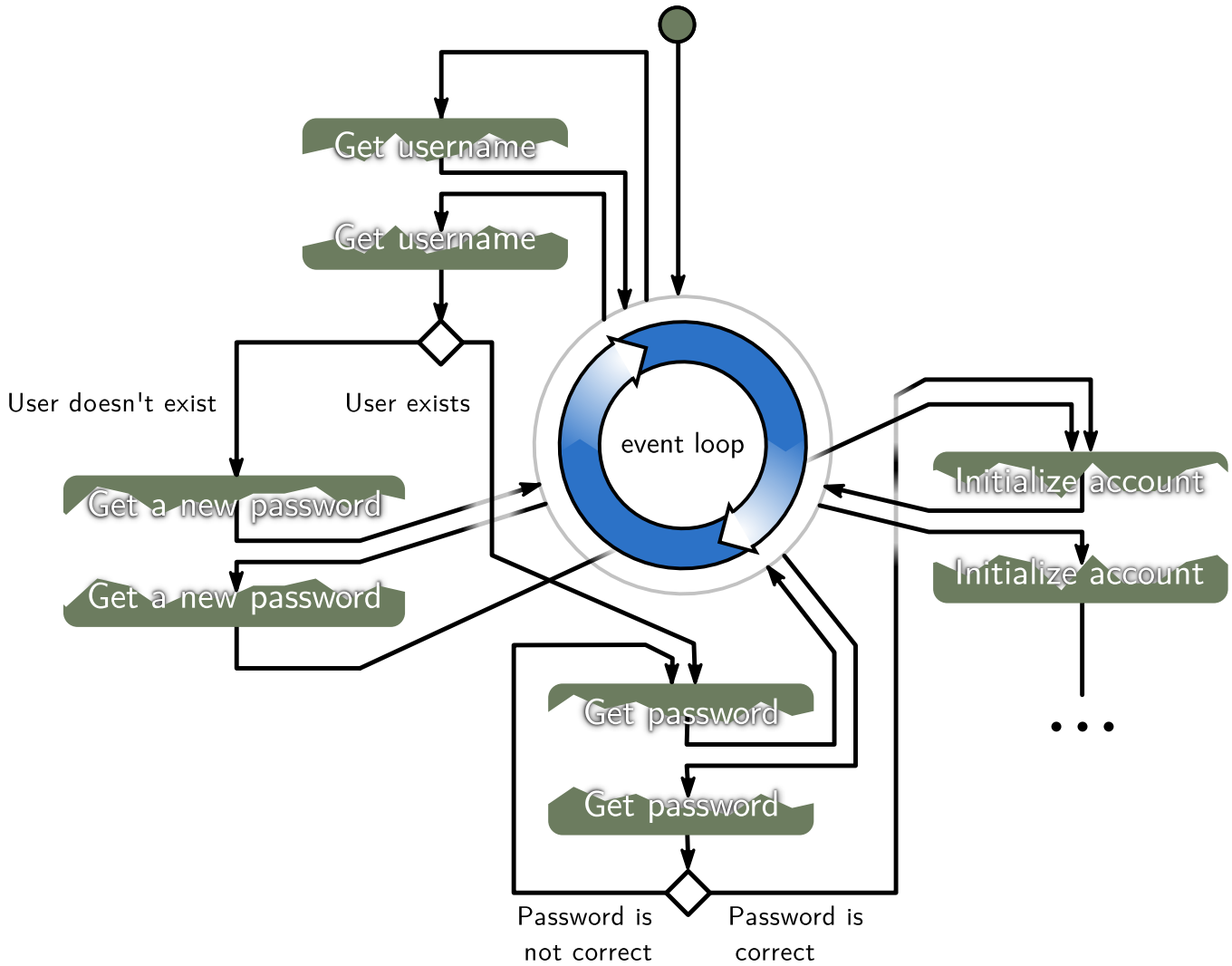



```
void login()
{
    user = get_username();

    new_user = !check_if_user_exists(user);

    if (new_user) {
        pass = get_password();
        initialize_account(uame, pass);

    } else do {
        pass = get_password();

    } while (!check_user(user, pass));

    initialize_environment();

    if (new_user) show_welcome_message();
}
```

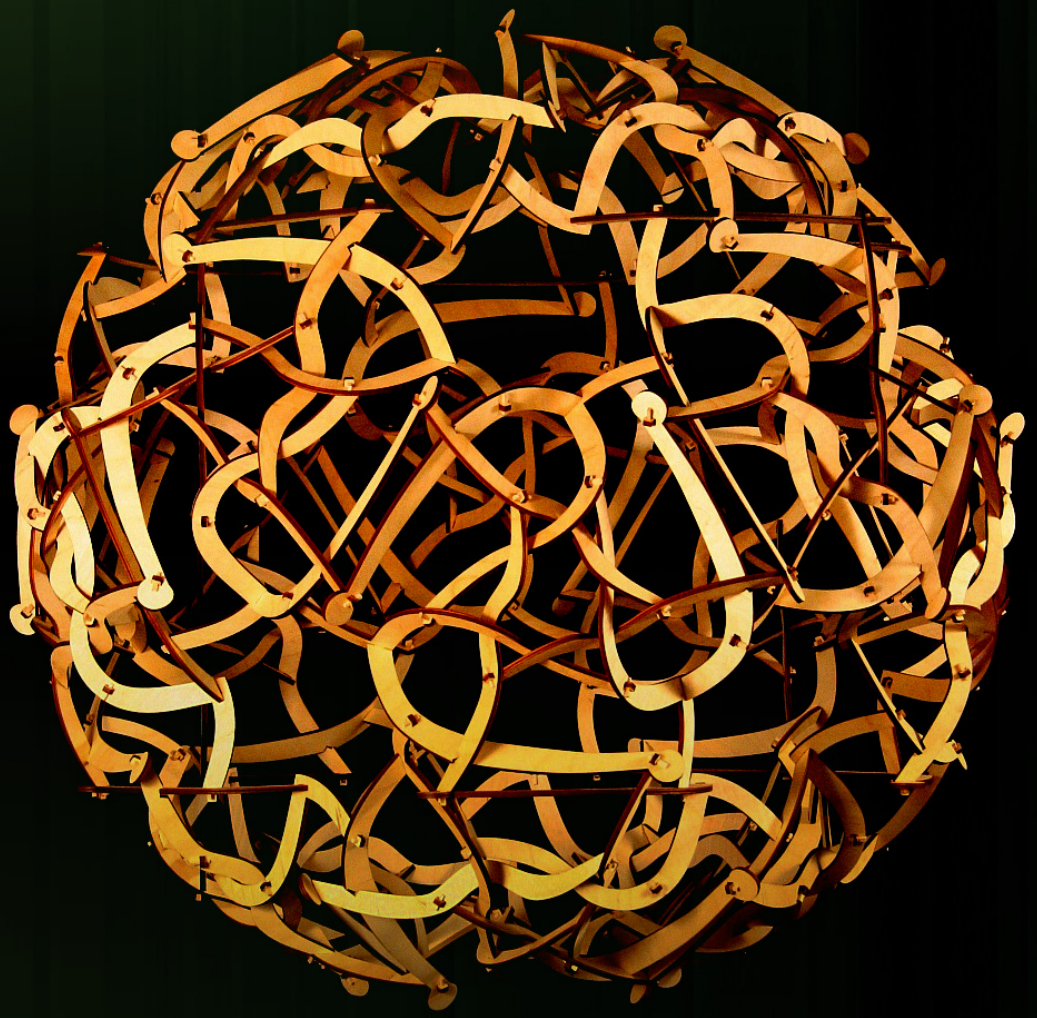
MEET JEFF

```
void login() { get_username(on_got_username); }

void on_got_username( ::: ) {
    new_user = !check_if_user_exists(user);
    if (new_user) {
        get_password(on_got_password);
    } else { ::: }
}

void on_got_password( ::: ) {
    check_user(user, password, on_user_checked);
}

void on_user_checked( ::: ) {
    if (!user_valid) {
        on_got_username(user);
    } else {
        initialize_environment(on_environment_initialized);
    }
}

:::
```

OUT OF CONTROL

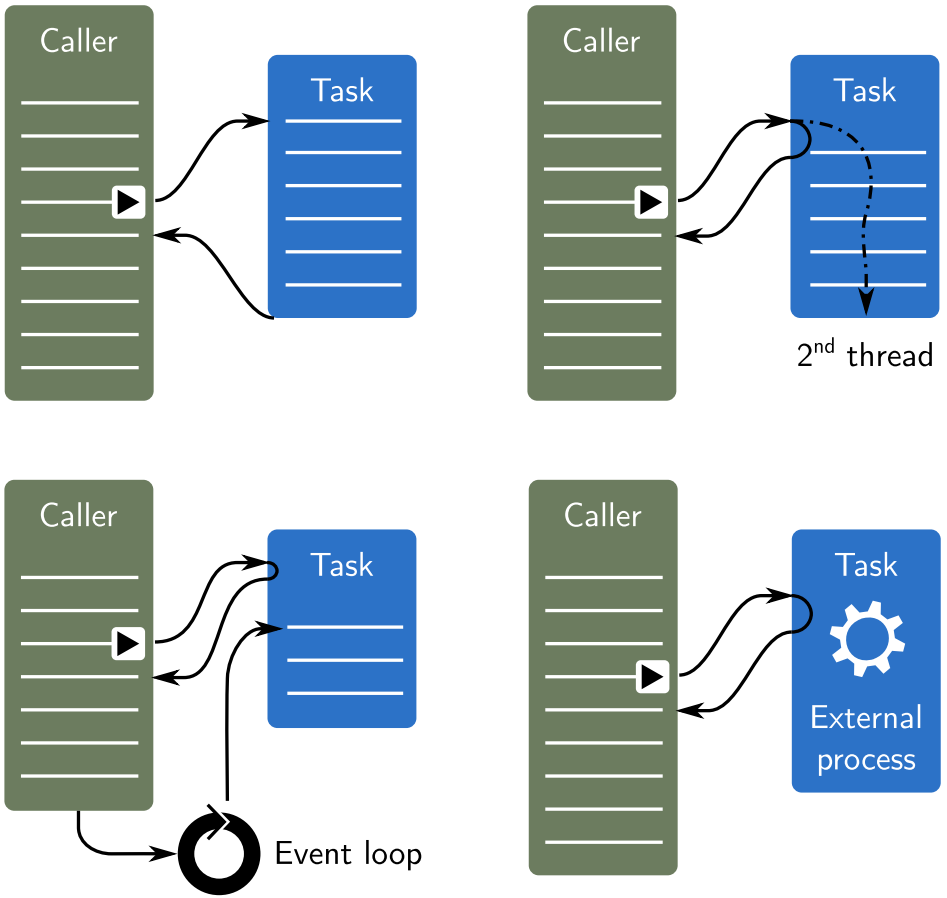


OUT OF CONTROL



"Spaghetti code" by George W. Hart

REASONS FOR WAITING



- User input
- Network actions
- Inter-process communication
- External process execution
- Communication with a slow database
- CPU-intensive work
- Heterogeneous computing

...

HIDEAWAY

- Wrapping it in task objects (`QThread`, `KJob`, …)
- Methods with time-outs (`select`, …)
- … or with validity checks (`QProcess::state`, …)
- Future values (`future<T>`, `QFuture<T>`, `QDBusPendingReply<T>`, …)

CONTINUATIONS

Lost in the Future

Under wraps

LOST IN THE FUTURE

- Is it about monads?
- Callbacks?
- Signals and slots?

LOST IN THE FUTURE

C++ standard proposal N3558, Boost.Thread 1.55.0

```
future<int> result = deepThought.meaningOfLife();

#if 0
    // this would block
    cout << result.get();
#endif

result.then([] (future<int> result) {
    // called when the result is available
    // call to .get() does not block here
    cout << result.get();
});
```

LOST IN THE FUTURE

```
int i;                          |  i.then(c);          // ERROR!
                                |
future<int> future;             |  future.then(c);     // ok
                                |
QFuture<int> qfuture;           |  qfuture.then(c);    // ERROR!
                                |
                                |
                                |
                                |
                                |
                                |
                                |
                                |
KJob *job;                      |  job->then(c);       // ERROR!
                                |
                                |
                                |
                                |
                                |
                                |
```

LOST IN THE FUTURE

```
int i;

future<int> future;

QFuture<int> qfuture;
```

```
c(i);

future.then(c);

auto watcher = new QFutureWatcher<int>();
QObject::connect(watcher,
    &QFutureWatcherBase::finished,
    [=] {
        c(watcher->result());
        watcher->deleteLater();
    });
watcher->setFuture(qfuture);
```

```
KJob *job;
```

```
QObject::connect(job,
    &KJob::finished,
    [] (KJob *job) {
        c(job-> ... something ...);
        job->deleteLater();
    });
```

UNDER WRAPS

```
template <typename _Job, typename _Continuation>
void continue_with(_Job &&job, _Continuation &&continuation)
{
    using is_nullary =
        typename std::is_constructible<
            std::function<void()>,
            _Continuation
        >::type;

    _continue_with_helper(
        job(),
        std::forward<_Continuation>(continuation), is_nullary()
    );
}
```

UNDER WRAPS

```
template <typename _ReturnType, typename _Continuation>
void _continue_with_helper(const _ReturnType &value,
                           _Continuation &&continuation,
                           std::true_type)
{
    continuation();
}

template <typename _ReturnType, typename _Continuation>
void _continue_with_helper(const _ReturnType &value,
                           _Continuation &&continuation,
                           std::false_type)
{
    using is_callable = ...;
    static_assert(is_callable::value,
        "The continuation needs to have zero or one argument");

    continuation(value);
}
```

UNDER WRAPS

```
template <typename _ReturnType, typename _Continuation>
void _continue_with_helper(const QFuture<_ReturnType> &future,
                           _Continuation &&continuation,
                           std::false_type)
{
    if (!future.isFinished()) {
        auto watcher =
            new QFutureWatcher<_ReturnType>();

        QObject::connect(watcher, &QFutureWatcherBase::finished,
            [=] {
                continuation(watcher->result());
                watcher->deleteLater();
            });

        watcher->setFuture(future);

    } else continuation(future.result());
}
```

MATCHBOX

```
template<typename _TestType, typename _ArgType>
class has_then_method {
private:
    template<typename U, void (U::*)(_ArgType)>
    struct test_struct {};

    template<typename U>
    static std::true_type test(test_struct <U, &U::then> *);

    template<typename U>
    static std::false_type test(...);

public:
    using type = decltype(test<_TestType>(nullptr));
    static const bool value =
        std::is_same<type, std::true_type>::value;
}
```

SCHEDULERS

The Chains are On

The New Order

Set Your Controls for the Heart of the Sun

THE CHAINS ARE ON

```
getUsername().then(
    [] (future<string> username) {
        getPassword().then(
            [=] (future<string> password) {
                createAccount(username, password).then(
                    ...
                );
            }
        );
    }
);
```

Localized, but still not readable. Can it be made nicer?

THE CHAINS ARE ON

Can it be made to look like this?

```
void login()
{
    ...
    username = getUsername();
    password = getPassword();
    createAccount(username, password);
}
```

No, but ...

THE CHAINS ARE ON

... it could look like this:

```
auto login = serial_
(
    ...
    username = getUsername(),
    password = getPassword(),
    createAccount(username, password)
);
```

Peculiar syntax, but much more readable.

THE NEW ORDER

```
template <typename... _Jobs>
class Serial;

template <>
class Serial<> : public QObject
               , protected QFutureInterface<int> {
public:
    ~Serial() {}

    int operator()()
    {
        reportResult(0);
        reportFinished();
        return 0;
    }
};
```

THE NEW ORDER

```
template <typename _Job, typename... _Jobs>
class Serial<_Job, _Jobs...> : public Serial<_Jobs...> {
private:
    using tail_t = Serial<_Jobs...>;
public:
    Serial(_Job &&job, _Jobs &&... jobs)
        : tail_t(std::forward<_Jobs>(jobs)...)
        , m_job(std::forward<_Job>(job)) {}

    QFuture<int> operator()() {
        auto future = this->future();

        continue_with(std::ref(m_job), [&] {
            tail_t::operator()();
        });

        return future;
    }

private:
    _Job m_job;
};
```

LET THERE BE MORE LIGHT

- while loop:

```
while_(                          |    while_(condition) (
    condition,                   |        body
    body                         |
)                                |    )
```

- branching:

```
if_(                             |    if_(condition) (
    condition,                   |        then_branch
    then_branch,                 |    ).else_(
    else_branch                  |        else_branch
)                                |    )
```

LET THERE BE MORE LIGHT

- asynchronous assignment

```
var<int> value;

value = 5;              // immediate assignment
value = someFuture(); // asynchronous assignment
```

- parallel execution

```
parallel_(
    task1,
    task2,
    ...
)
```

- parallel without waiting

```
detach_(task)
```

- producer-consumer
- transactions

...

SET YOUR CONTROLS...

```
var<int> wait;

serial_(
    test::writeMessage(0, "Starting the program"),

    wait = test::howMuchShouldIWait(7),
    test::writeMessageAsync(wait,
            "What is the answer to the Ultimate Question of Life, "
            "the Universe, and Everything?"
        ),

    while_(test::howMuchShouldIWait(0),
        test::writeMessageAsync(1, "42")
    ),

    serial_(
        test::writeMessageAsync(1, "We are going away..."),
        test::writeMessageAsync(1, "... sorry, but we have to.")
    ),

    test::writeMessage(0, "There, you have it!")
)();
```

... FOR THE HEART OF THE SUN

```
while_(
    // Wait until we get a connection.
    client = ws::server::accept(server),

    // Start a detached execution path to process the client.
    detach_([] {
        var<ws::client_header> header;
        var<ws::message> message;
        var<string> server_key;

        serial_(
            // WebSocket handshake
            header = ws::client::get_header(),
            server_key = ws::server::create_key(header),
            ws::client::send_header(client, server_key),

            // Sending the initial greeting message
            ws::client::message_write(client, "Hello, I'm Echo"),

            // Connection established
            while_(
                // getting and echoing the message
                message = ws::client::message_read(client),
                ws::client::message_write(client, message)
            )
        )
    })
)
```

TASKS

Lazy Day

LAZY DAY

Problem:

A method is executed while the arguments are evaluated.

```
serial_(
    someMethod(0, "Starting the program"),
    ...
);
```

`someMethod` must not do anything, but return a functor.

LAZY DAY

So, your options are:

```
void someMethod(...);

serial_(
    std::bind(someMethod, 0, "Starting the program"),
    ...
)
```

LAZY DAY

Or using a std::bind-based wrapper

```
namespace detail {
    void someMethod(...);
}

auto someMethod(...)
    -> decltype(std::bind(detail::someMethod,
                std::forward arguments ...))
{
    return std::bind(detail::someMethod,
                     std::forward arguments ...);
}

serial_(
    someMethod(0, "Starting the program"),
    ...
)
```

LAZY DAY

Or using a simple wrapper:

```
namespace detail {
    void someMethod(...);
}

auto someMethod = curry(detail::someMethod);

serial_(
    someMethod(0, "Starting the program"),
    ...
)
```

EPILOGUE

Benefits:

- Readable code, easy to reason about
- Automatic lifetime management
- Advanced control structures compared to plain C++

Things to get used to:

- Peculiar syntax
- Some functional programming constructs like purity, immutability, etc. are preferred
- Less expressive statements

ANSWERS? QUESTIONS! QUESTIONS? ANSWERS!

Kudos:

- Friends at KDE
- Dr Saša Malkov
- KDAB
- basysKom
- LaTeX and Beamer developers