Example

```
def _activate_in_arta(subscription):
    """
    ...
    """
    if not gargoyle.is_active('arta:provisioning:activations'):
        return False

    # Create backendcli and contact in Arta if necessary
    arta_contact_already_exists = False
    try:
        cli = subscription.backendcli
        if cli.contactid:
            arta_contact_already_exists = True
    except BackendCLI.DoesNotExist:
        cli = BackendCLI.objects.create(subscription=subscription)
    if not arta_contact_already_exists:
        if not create_arta_contact(subscription):
            return False
        # Needs to be 'refreshed' to get the correct value
        cli = subscription.backendcli

    # Get the SN that's provisioned on the SIM
    if cli.sn.strip() == '':
        cli.sn = get_sn_for_sim(subscription.mobilenumber.active_sim)
        cli.save()
        ...
```

Example

```
...
# Push the CLI to the next state if necessary
state = State.objects.default(cli)
if state is None:
    return False
elif state.current_is['CLI_SIM_ACTIVE']:
    result = True
elif state.current_is['NOT_IN_ARTILIUM']:
    result = state.action(None, 'attach_cli_to_user')
elif state.in_group['can_resume']:
    result = state.action(None, state.get_action_names()[0])
else:
    return False

# Set all warning SMS as SENT
for f in Flag.objects.filter(name__startswith='ALERT', name__endswith='SENT'):
    f.add_subscription(subscription)

if result:
    if not subscription.has_number_porting:
        Status.objects.sim_activated(subscription)
    else:
        Status.objects.number_porting_requested(subscription)

    # mark sim as active
    act = Activation.objects.get(subscription=subscription)
    act.activated_on = datetime.now()
    act.save()
    ...
```

Example

```
...
    # Put all topups on hold back to paymentdone
    subscription.topup_set.get_on_hold().update(status=TOPUP_STATUS_PAYMENT_DONE)
else:
    from mvne.activation.utils import send_activation_failure_email
    send_activation_failure_email(subscription)
    return False

return True
```

Issues

- Function is long ($>$15 lines)
- Function intent is clear by name, but details of function requires effort understand
- Code is separated in logical chunks but no real abstraction mechanism is used

My beef with comments

- Will get outdated when code changes
- Gives wrong/incomplete information
- Often used as an alternative readable code
- Less thought given to good naming

Example

```
def _activate_in_arta(subscription):
    """
    ...
    """
    if not gargoyle.is_active('arta:provisioning:activations'):
        return False

    cli = get_or_create_backend_cli()

    set_sn_on_sim(subscription, cli)
    next_state = push_cli_to_next_state(cli)
    notify_sim_sent(subscription)
    activate_subscription(subscription)
```

Benefits

- Content of function is all at the same level of detail.
- Easier to understand the function in question. More details can be found for each separate step
- Forces more thought through names (no one wants super long names, but intent needs to be clear)
- Incentives functional style (which is clearer IMO)
- Disincentives state change in between logical chunks