Describing features

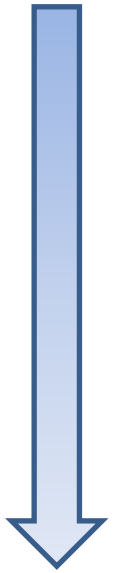
Computing & Information Sciences

W. H. Bell

Software development lifecycle

One iteration

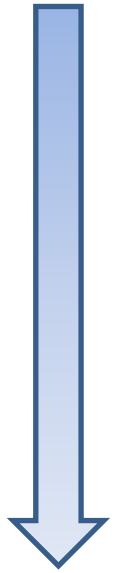
- Requirements definition.
- **Software and systems design.**
- Implementation and unit testing.
- Integration and system testing.
- Operation and maintenance.



System description

University of Strathclyde Glasgow

- User interface design.
  - Final design loosely coupled to framework choice.
- Data model.
  - Final implementation coupled to data flow/serialisation choices.
- Architecture.
  - Affects how software is implemented.
- **Describing functionality.**
  - Software agnostic, but expressed within architecture.



High and low-level design

University of Strathclyde Glasgow

- High-level design.
  - Architecture.
  - Services.
  - Generalised design of large components.
- Low-level design.
  - Internal APIs.
  - Internal processes.
  - Classes and implementation overview.

Cost vs benefit

University of Strathclyde Glasgow

- Design effort matched to project.
  - V-lifecycle – larger static document collection.
  - Agile – smaller document collection, updated during development.
- Design is costly.
  - Focus on main components of high-level design.
  - Limit low-level design details.
  - Document implementation.
- Lack of design is costly.
  - Developers may produce divergent changes.

UML use-case diagrams

University of Strathclyde Glasgow

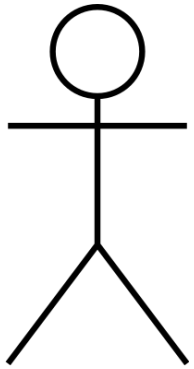
- High-level design tool.
  - Used to understand user interactions with software.
- Enable discussion and requirements elicitation.
  - During requirements capture.
  - To improve requirements description.
  - When designing user acceptance tests.

UML use-case diagrams

University of Strathclyde Glasgow

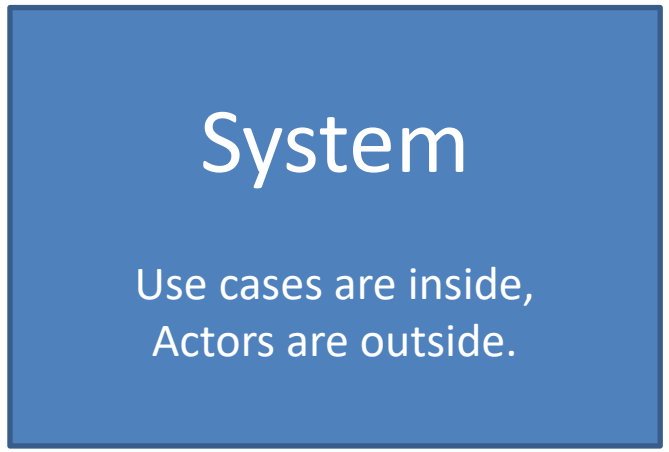
- Use cases
  - Function provided by the system – needed by user.
- Actors
  - A user role, which interacts with system.
  - Person, organisation or persona (computer/external system).
- Subsystems
  - Used to represent large-scale components within application.
  - Common to class and use-case diagrams.
- Relationships

UML use-case diagrams

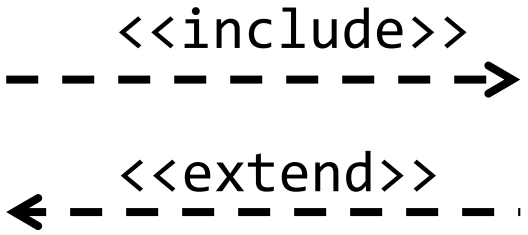


Actor

Relationship
(Actor and user case)



Relationship
(Use cases)

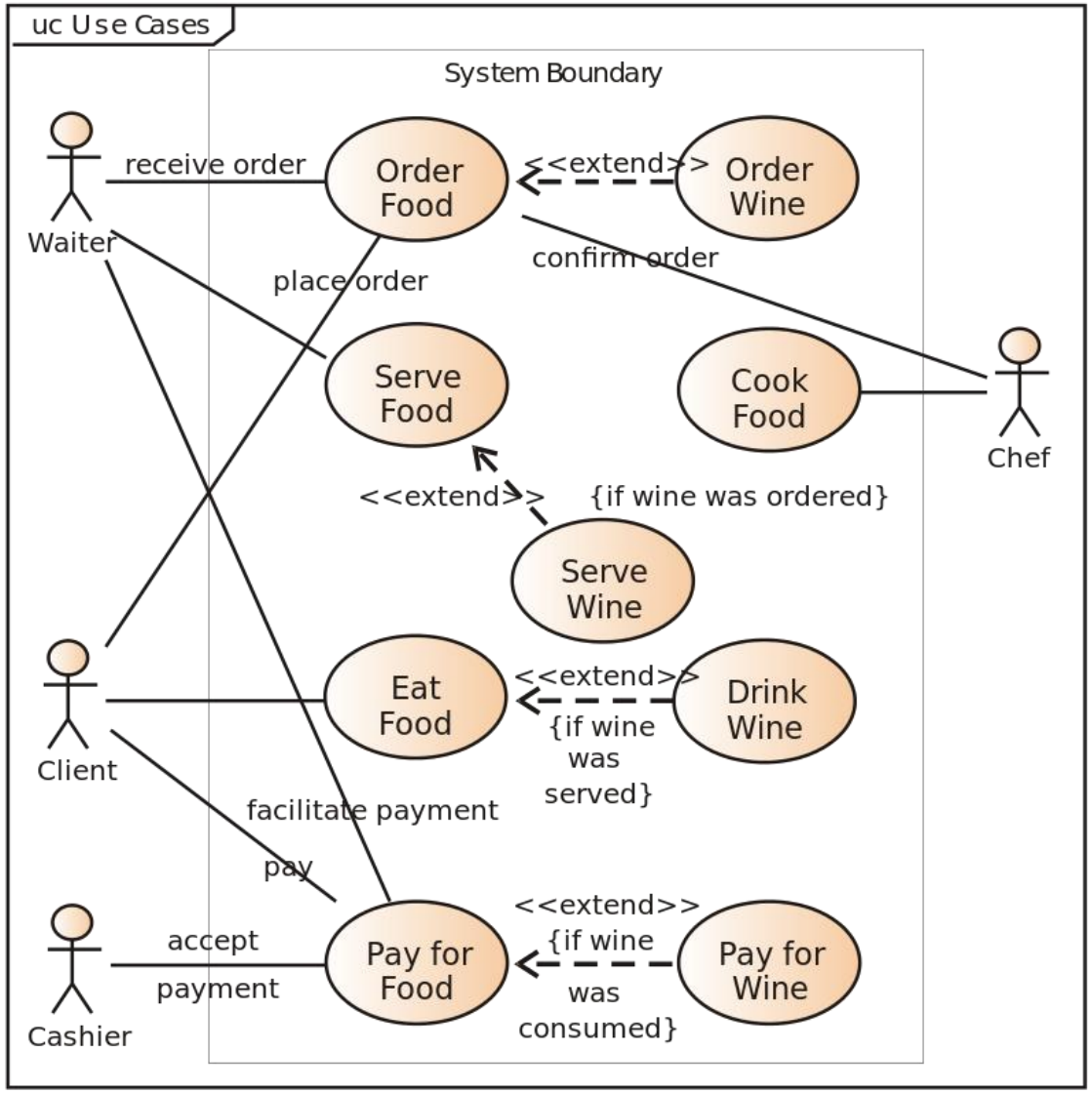


UML use-case diagrams

University of Strathclyde Glasgow

- Extension (`<<extend>>`).
  - Extended from another use case.
  - Base use case is optional.
  - Can include condition within diagram.
- Inclusion (`<<include>>`).
  - Includes another use case.
  - Base use case is required.

UML use-case diagrams



https://en.wikipedia.org/wiki/Use_case_diagram

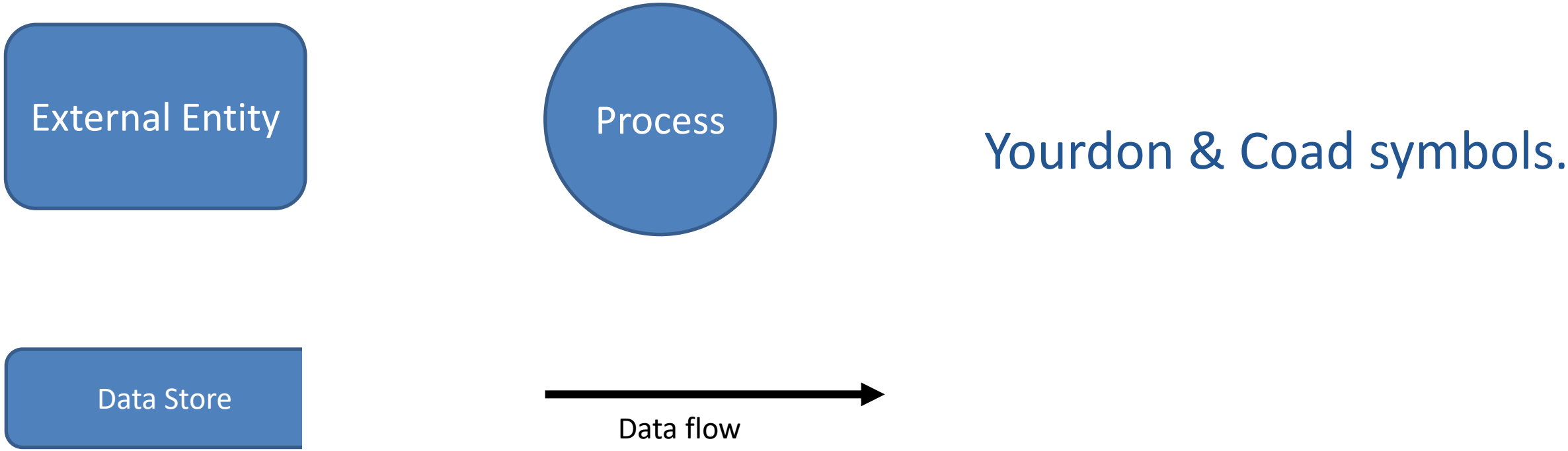
Data flow diagram

University of Strathclyde Glasgow

- Data flow graph.
  - External entity – sends or receives data.
  - Process – changes the data.
    - Number processes to describe sequences.
  - Data store – holds information.
  - Data flow – between external entities, processes and stores.
- With or without object-oriented design.

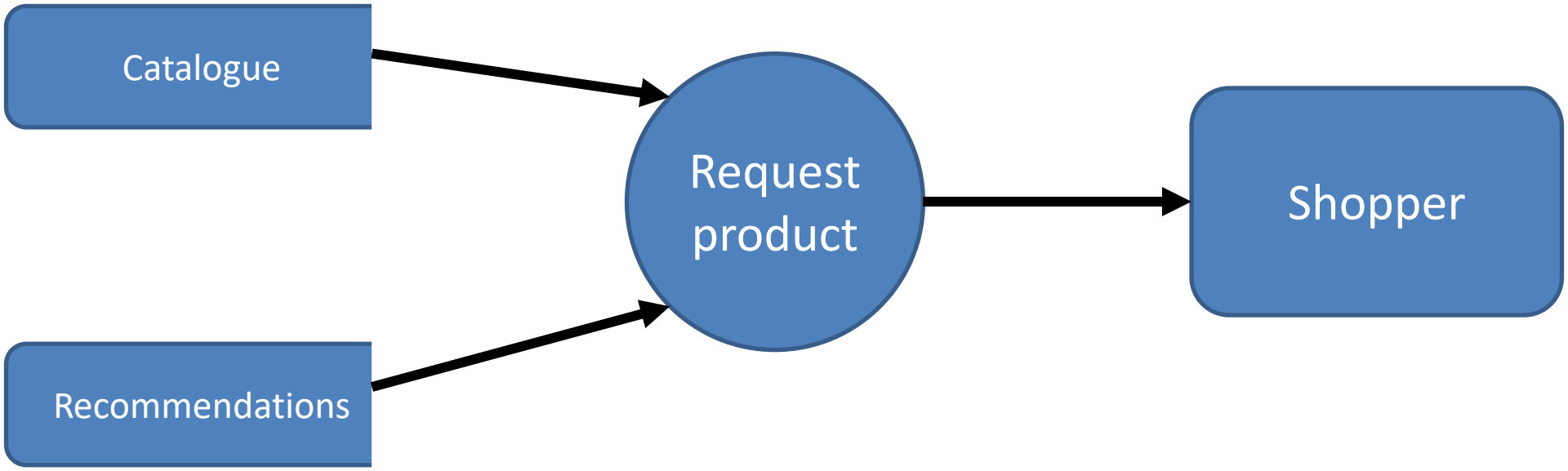
L. L. Constantine, E. Yourdon, “Structured design, Fundamentals of a Discipline of Computer Program and Systems Design”, 1978, Yourdon Press

Data flow diagram (DFD)



Yourdon & Coad symbols.

Data flow diagram



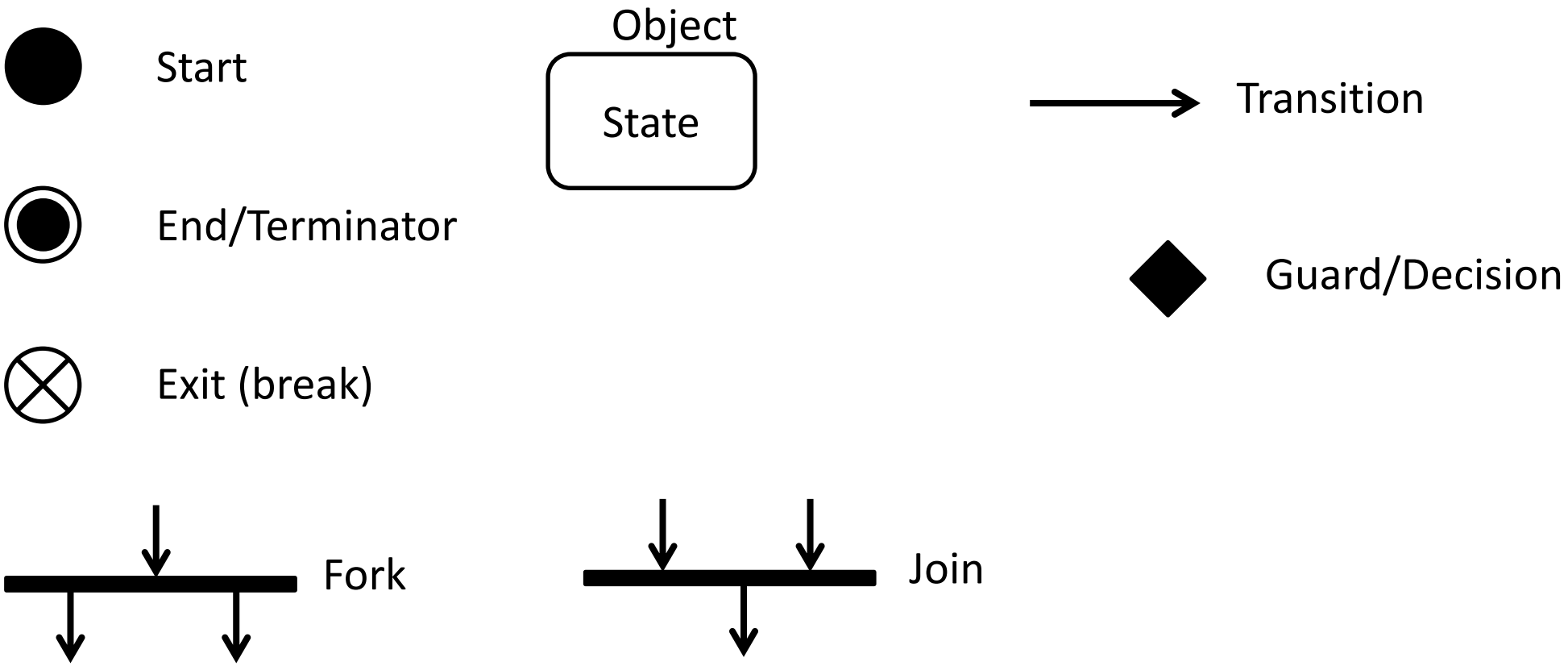
DFD Levels

- Context diagram.
- Level 0 – expand system process.
- Level 1 – expand selected process.

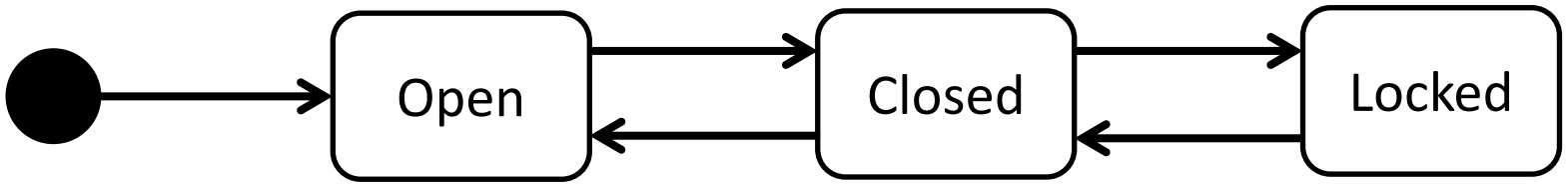
State diagram

- Describe state of object.
- Arrows from one state to another.
  - Condition descriptions next to lines – optional
  - Loop back to same state.
- Levels – expand state into sub-diagrams.

UML state diagram



UML State diagram: door example



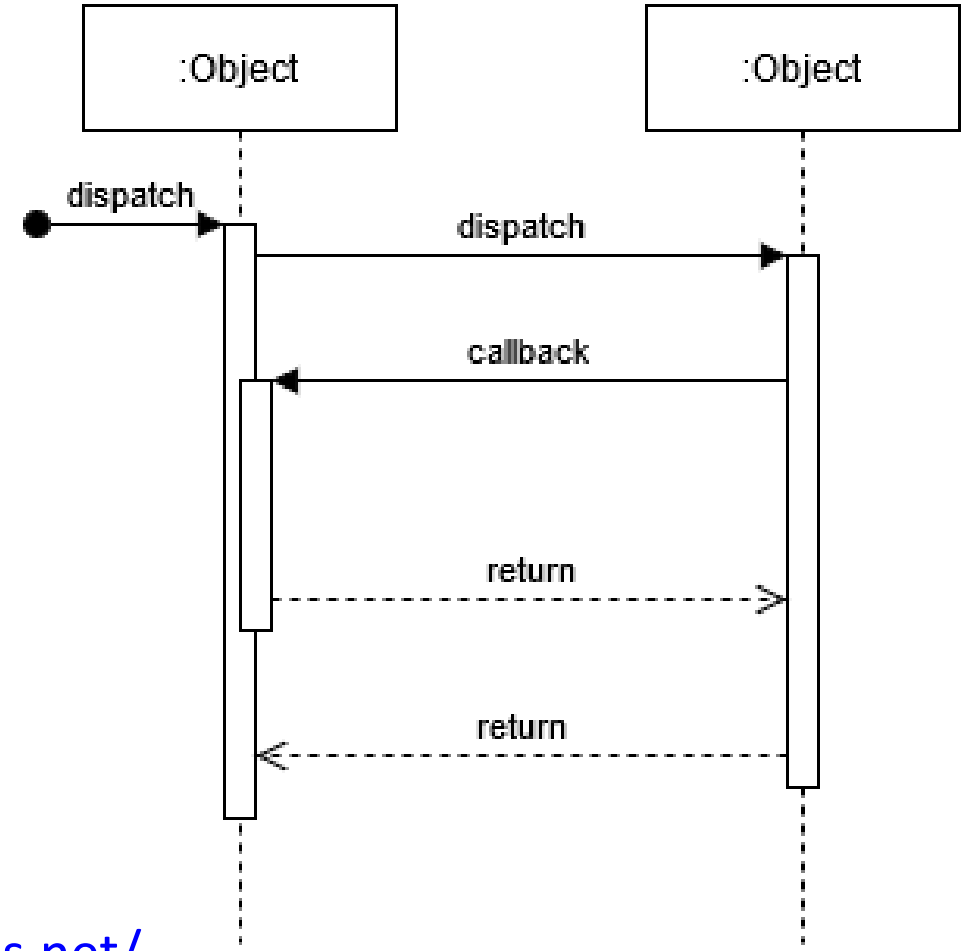
https://sparxsystems.com/resources/tutorials/uml2/state-diagram.html

UML sequence diagrams

University of Strathclyde Glasgow

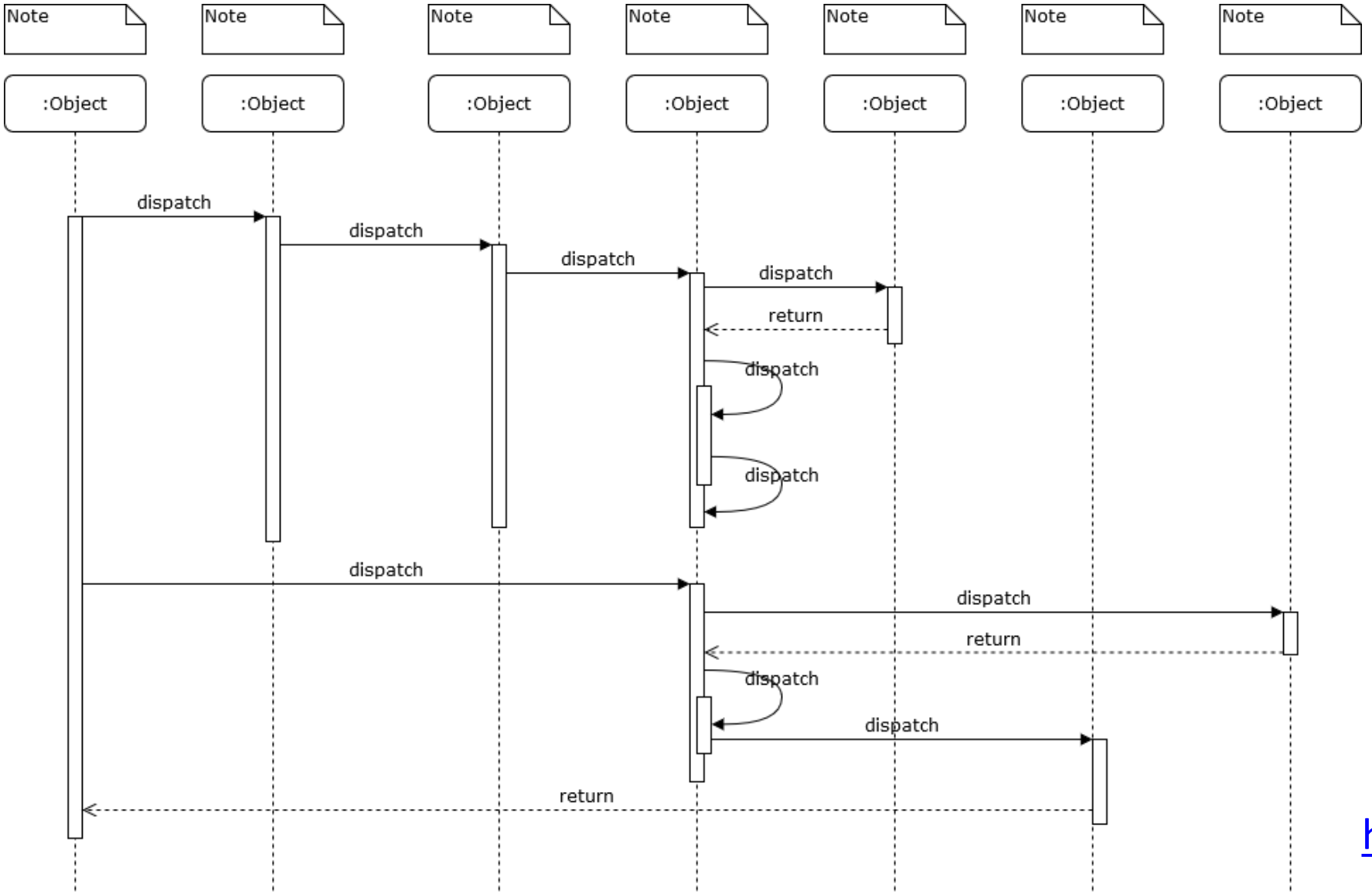
- Describe interactions between application layers.
  - Client and server.
  - Layer architecture.
  - API calls.
  - Internal function calls.
- UML syntax includes loops.
  - Typically, limit documented details due to cost.

UML Sequence Diagram



https://app.diagrams.net/

UML Sequence Diagram



Need to limit size of diagram.

https://app.diagrams.net/

Class responsibility collaborator (CRC)

University of Strathclyde Glasgow

- Used for object-oriented design.
  - Can map objects to relational tables, using normalisation.
- Used as a brainstorming technique.
  - Followed by sequence diagrams – capturing interactions.

Class Responsibility Collaborator (CRC) model (Beck & Cunningham 1989; Wilkinson 1995; Ambler 1995)

CRC Cards

University of Strathclyde Glasgow

Class Name	
Responsibilities	**Collaborators** (Other classes)

http://agilemodeling.com/artifacts/crcModel.htm

Class Responsibility Collaborator (CRC) model (Beck & Cunningham 1989; Wilkinson 1995; Ambler 1995)

CRC Cards

University of Strathclyde Glasgow

Class Name: Customer	
Responsibilities: Places orders Knows name Knows address Knows customer number Knows order history	**Collaborators:** Order

http://agilemodeling.com/artifacts/crcModel.htm

Class Responsibility Collaborator (CRC) model (Beck & Cunningham 1989; Wilkinson 1995; Ambler 1995)

Control-flow graph

University of Strathclyde Glasgow

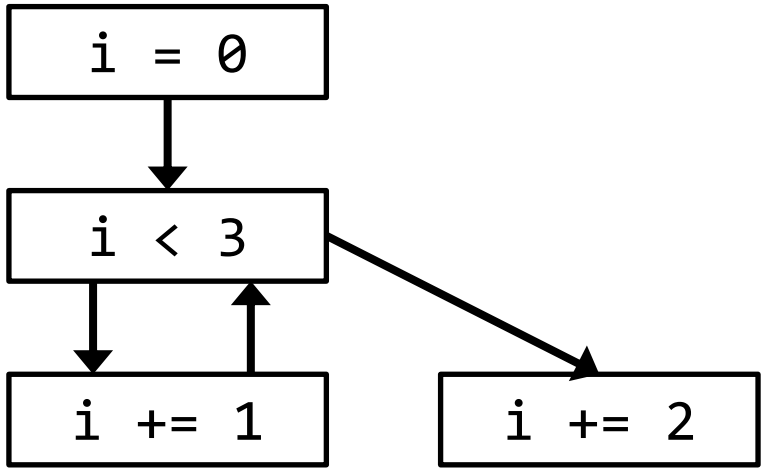
- Describe low-level functionality.
  - Detailed logic and decisions.
- Nodes – assignment and functions.
  - No conditions and loops.
- Arrows – conditions and loops.

- Document implementation.
  - Label implementation with integers.
  - Use labels within nodes.

F. E. Allen, “Control Flow Analysis”, IBM Corporation, 1970

Control-flow graph

```
i = 0
while i < 3:
    i += 1
i += 2
```



F. E. Allen, “Control Flow Analysis”, IBM Corporation, 1970

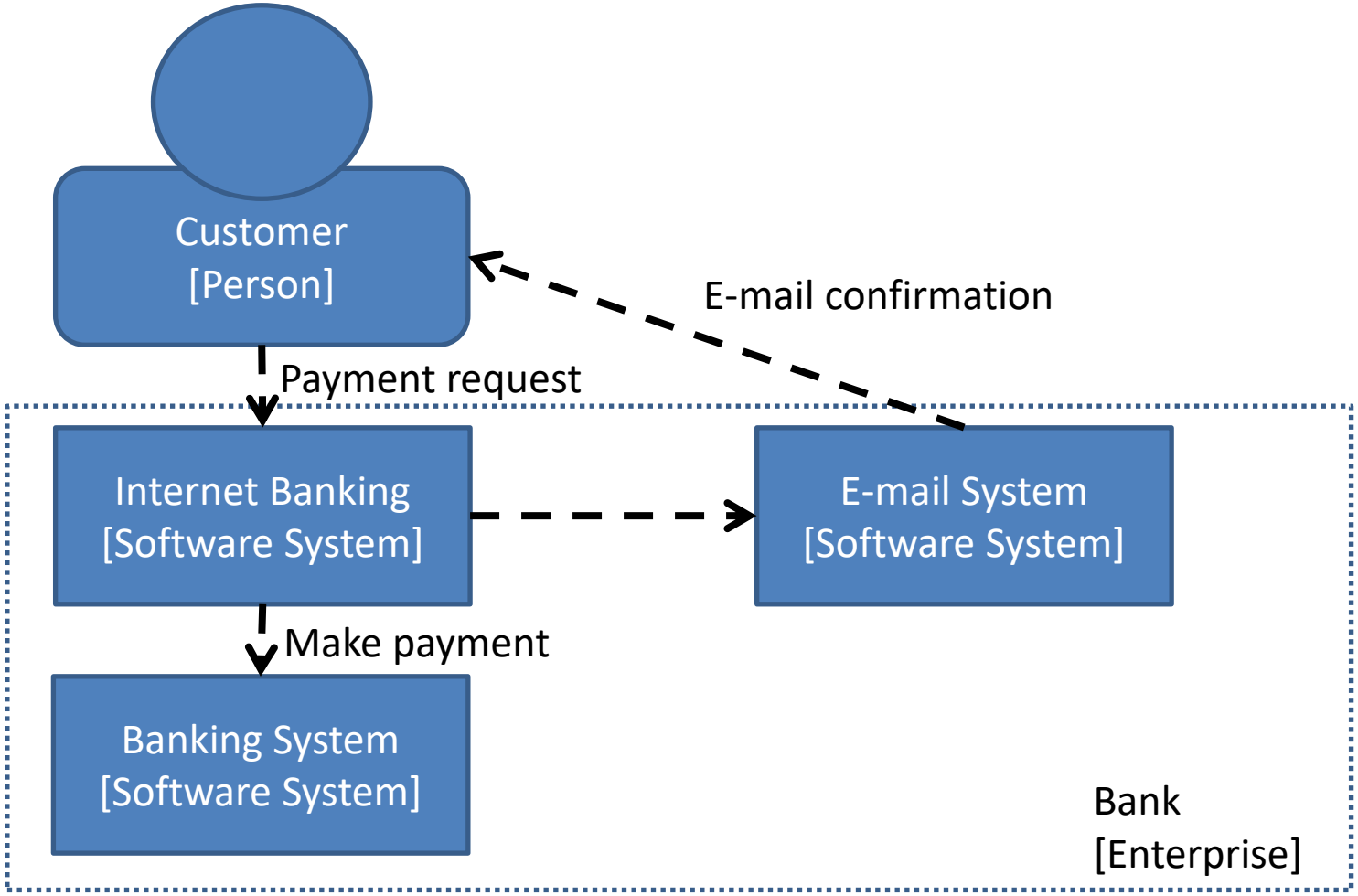
C4 model

University of Strathclyde Glasgow

- Use architectural view points.
  - Context diagrams – system, users and other users.
  - Container diagrams – context as interrelated containers.
    - Application or data store.
  - Component diagrams – containers as interrelated components.
  - Code diagrams – UML class diagram, entity relation diagrams.

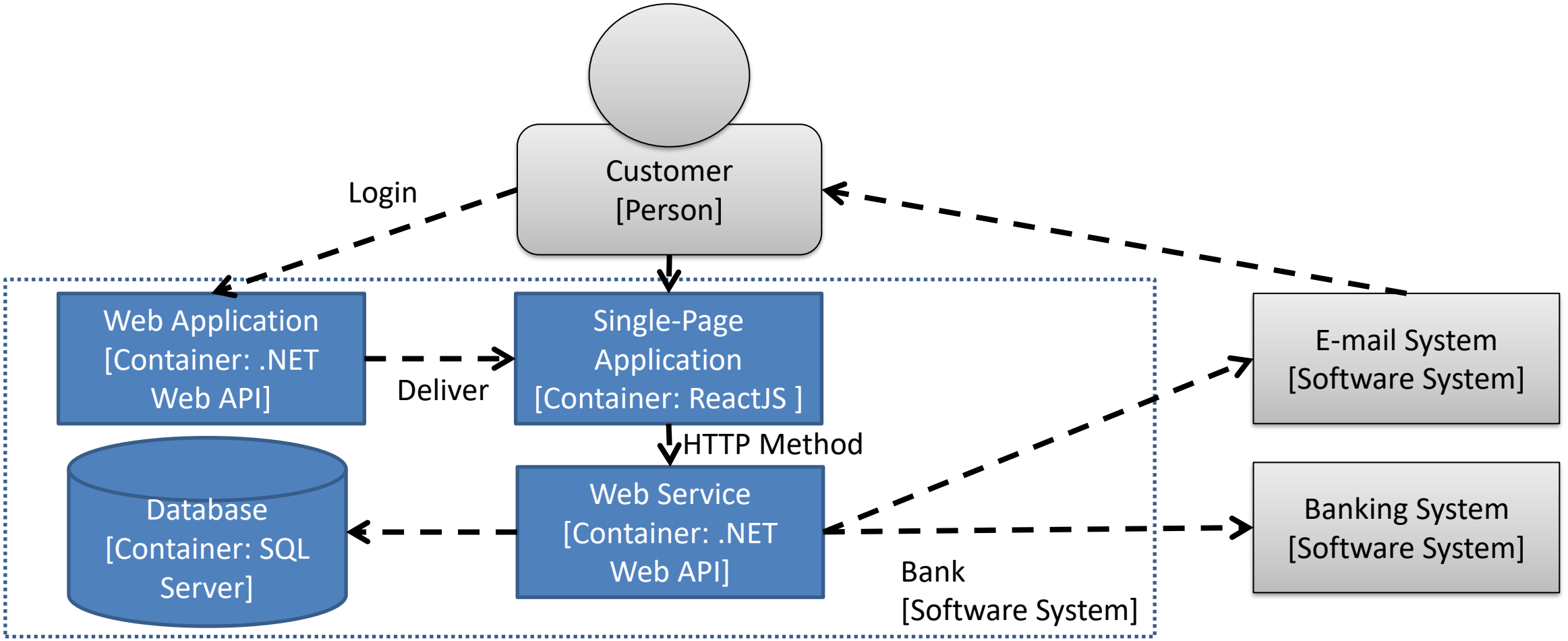
https://c4model.com/

C4 model: context



https://c4model.com/

C4 model: container



https://c4model.com/

University of Strathclyde Glasgow

Design patterns

Design patterns

University of Strathclyde Glasgow

- Associated with object-oriented design.
  - Originally produced for Smalltalk and C++.
  - Partially applicable to newer languages.

E. Gamma, et al. “Design patterns: elements of reusable object-oriented software”, Pearson Addison Wesley, 1994

Common design patterns

University of Strathclyde Glasgow

- Decorator.
  - Attach additional responsibilities to object dynamically.
- Factory method.
  - Interface for creating an object.
- Iterator.
  - Access elements of series of objects.

E. Gamma, et al. “Design patterns: elements of reusable object-oriented software”, Pearson Addison Wesley, 1994

Common design patterns

University of Strathclyde Glasgow

- Singleton.
  - Once instance of a class only.
- Template method.
  - Implemented with one or more types.

E. Gamma, et al. “Design patterns: elements of reusable object-oriented software”, Pearson Addison Wesley, 1994

Decorator

University of Strathclyde Glasgow

- Often used within web app frameworks.
  - .NET MVC.
  - Python Flask.
- Framework reads decorators to form mapping.
  - Input URL mapped to function.

Decorator: Python Flask

University of Strathclyde Glasgow

Prefixed by “@” character.

```
books_bp = Blueprint('books', __name__, url_prefix='/books')


@books_bp.route('/', methods=['GET', 'POST'])
def books():
    if request.method == 'GET':
        return jsonify([book.to_dict() for book in models.Book.query.all()])
```

Not completely consistent with original “decorator” definition.

https://docs.python.org/3/glossary.html#term-decorator

Factory

University of Strathclyde Glasgow

- Used for services.
  - Each object run with separate thread.
  - Allow hosting to generate configurable threads.

Factory: Python Flask

University of Strathclyde Glasgow

```
def create_app(test_config:dict = {}):
    app = Flask(__name__)

    app.config['SQLALCHEMY_DATABASE_URI'] = database_uri()
    app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
    if len(test_config) > 0:
        app.config.update(test_config)
    with app.app_context():
        db.init_app(app)
        db.create_all()
    from routes import books_bp
    app.register_blueprint(books_bp)
    return app
```

https://flask.palletsprojects.com/en/2.2.x/patterns/appfactories/

Singleton

University of Strathclyde Glasgow

- Control data or state access.
  - Configuration data from one source/cache.
  - One service instance/class to handle requests.
- Implemented with private constructor.
  - Static variable contains reference to single instance.
- Language specific.
  - Many objected-oriented languages, but not Python.
  - Python singleton can be constructed with a module.

https://www.digitalocean.com/community/tutorials/java-singleton-design-pattern-best-practices-examples

Conclusions

- Design approaches facilitate low-level design.
  - Must balance cost vs benefit and target key areas.
  - Brainstorming with light-weight techniques.
- Diagram standards support feature design.
  - Facilitate discussions with clients.
  - Limit information to avoid endless design loops.
- Choose appropriate design patterns.

University of Strathclyde Glasgow