Component-Level Design (Chapter 14 – Software Engineering: A Practitioner’s Approach)

Summary:

- **Component-level design focuses on defining how each software component works, moving from abstract architecture to implementable code.**
- **It stresses cohesion (each component doing one job well) and low coupling (minimal interdependence).**
- **Introduces design classes: interface classes (UI), controller classes (logic), and entity classes (data).**
- **Uses Structured Design and Object-Oriented Design approaches to refine software modules.**
- **Discusses design heuristics such as:**
    - **Keep interfaces small and simple.**
    - **Separate concerns.**
    - **Anticipate change via design flexibility.**
- **Encourages component reuse and design patterns (e.g., MVC, Observer, Factory). Concludes with quality attributes for components: reliability, maintainability, and testability.**

✅ Ideal for developers needing to translate software architecture into class/method-level planning.

Focus: Moving from architecture to modular, code-level structure — *exactly the zone you’re in with ClinicTrends AI*.

🔍 Key Concepts and How They Relate to ClinicTrends AI:

Concept	Explanation	ClinicTrends AI Connection
Component-level design	Defines how each component should behave, interact, and be implemented.	You broke your app into: UI (Streamlit), Data Processor, NLP Engine, Alert System, etc. Each one is a component.

Design classes	Categorized into: Entity (data), Boundary (UI/API), and Control (logic) classes.	Your Data Processor is a control component; SurveyEntry or NPSRecord could be entity classes; Streamlit UI pages are boundary classes.
Cohesion	Each component should do one thing well.	Your NLP module does only text classification & topic modeling – great cohesion.
Coupling	Components should depend on each other minimally.	Your pipeline is modular – changing the NLP model doesn’t break the UI. That’s low coupling.
Heuristics	e.g., “Anticipate change”, “Use abstraction”, “Limit interaction to necessary interfaces”.	You applied this by using abstraction for file input and standardizing CSV schema validation.
Reusability	Components should be reusable in other contexts.	Your NPS predictor or BERTopic module could be reused in another dashboard or business unit (e.g., finance, marketing).
Quality attributes	Testability, Maintainability, Modularity, etc.	Your app benefits from testable pipelines (e.g., plug-in CSV, see results instantly), which supports maintainability.

How to improve further:

- Consider **formalizing contracts** between components with pydantic models or dataclasses (even inside Streamlit).
- If scaling: extract components into FastAPI microservices, maintaining modularity.