🚀 Hardware Team Slack Digest System - Complete Implementation

System Overview

A comprehensive Slack digest system that transforms overwhelming hardware team communications into actionable daily insights. Features multiple AI agents, GTM risk scoring, and a killer dashboard.

📋 Digest Structure

What the Hardware Team Sees Every Morning:

markdown

🏭 Hardware Team Daily Digest - Galaxy S26 Ultra Launch
Date: March 15, 2024 | **Days to Launch**: 47 | **GTM Risk Score**: 72/100 ⚠️

🎯 Today's Priorities
1. **[CRITICAL]** Resolve display supplier delay - Meeting @ 10 AM
2. **[HIGH]** Camera module QA sign-off needed by EOD
3. **[MEDIUM]** Review thermal test results from overnight run

🏆 Yesterday's Wins
- ✅ Battery life tests exceeded target by 15% (48hrs achieved!)
- ✅ FCC certification pre-approval received
- ✅ Production line efficiency improved to 94.5% OEE
- ✅ Alternative chip supplier qualified for backup

🚨 Blockers & Risks
- 🔴 **Display Supply Chain**: 3-day delay from Samsung Display
  - Impact: Could slip launch by 1 week
  - Action: Emergency supplier meeting scheduled
- 🟡 **Camera Autofocus**: 2.3% defect rate (target: <1%)
  - Impact: May require firmware update
  - Owner: @john_qa

📊 Key Metrics
Metric	Current	Target	Trend
Daily Production	9,847 units	10,000	↑ +3.2%
Defect Rate	1.8%	<2%	↓ -0.3%
Inventory Coverage	18 days	21 days	→
Supplier On-Time	91%	95%	↓ -2%

💬 Important Decisions Made
- **Approved**: Move to dual-supplier strategy for displays
- **Decided**: Implement additional QA checkpoint for camera modules
- **Postponed**: Packaging design change to next quarter

📝 Action Items by Team
Engineering
- [ ] Submit thermal management report - @sarah_eng (Due: 3 PM)
- [ ] Review antenna placement simulation - @mike_rf (Due: Tomorrow)

Supply Chain
- [ ] Negotiate expedited shipping for displays - @lisa_supply (Due: Today)
- [ ] Update component risk matrix - @raj_procurement (Due: EOW)

```
### Quality
- [ ] Root cause analysis on camera defects - @john_qa (Due: 5 PM)
- [ ] Prepare audit documentation - @amy_quality (Due: Thursday)

## 🔮 Looking Ahead
- **Tomorrow**: Design review for final prototype
- **This Week**: Start pilot production run (1000 units)
- **Next Week**: Regulatory submission deadline

## 💡 AI Insights
Based on current trends, recommend accelerating camera firmware development to mitigate quality
```

🤖 Complete Agent Architecture

1. Supply Chain Agent

```python
supply_chain_agent = Agent(
    name="Supply Chain Analyst",
    model="gpt-4",
    instructions="""You are a supply chain expert for smartphone manufacturing.
    Monitor and analyze:
    - Component availability and lead times
    - Supplier performance metrics
    - Inventory levels and safety stock
    - Geopolitical risks and disruptions
    - Cost fluctuations and negotiations

    Provide actionable recommendations for supply chain optimization.""",
    tools=[
        check_inventory_levels,
        analyze_supplier_performance,
        predict_shortage_risk,
        find_alternative_suppliers
    ]
)
```

2. Design Agent

```python
design_agent = Agent(
    name="Design Decision Tracker",
    model="gpt-4",
    instructions="""You are a hardware design specialist who tracks design changes.
    Focus on:
    - Design change requests and approvals
    - BOM (Bill of Materials) updates
    - CAD file revisions
    - Design validation test results
    - Cross-functional design impacts

    Identify how design decisions affect manufacturing and timeline.""",
    tools=[
        track_design_changes,
        analyze_bom_impact,
        check_design_validation,
        assess_manufacturability
    ]
)
```

3. Quality Agent

python

```
quality_agent = Agent(
    name="Quality Control Specialist",
    model="gpt-4",
    instructions="""You are a quality assurance expert for hardware manufacturing.
    Analyze:
    - Defect rates and patterns
    - Test results and validations
    - Compliance and certification status
    - RMA and failure analysis
    - Quality trending and predictions

    Provide early warning for quality issues.""",
    tools=[
        analyze_defect_patterns,
        predict_quality_issues,
        check_compliance_status,
        generate_quality_report
    ]
)
```

4. Timeline Agent

```python
timeline_agent = Agent(
    name="GTM Timeline Coordinator",
    model="gpt-4",
    instructions="""You are a project timeline expert for hardware launches.
    Track:
    - Critical path activities
    - Milestone achievements and delays
    - Cross-team dependencies
    - Resource allocation conflicts
    - Schedule compression opportunities

    Identify risks to launch timeline.""",
    tools=[
        analyze_critical_path,
        identify_dependencies,
        calculate_schedule_impact,
        suggest_mitigation_strategies
    ]
)
```

5. Production Agent

python

```python
production_agent = Agent(
    name="Production Monitor",
    model="gpt-4",
    instructions="""You are a manufacturing operations specialist.
    Monitor:
    - Production line efficiency (OEE)
    - Throughput and cycle times
    - Equipment status and maintenance
    - Shift performance comparisons
    - Capacity utilization

    Optimize production for launch readiness.""",
    tools=[
        get_production_metrics,
        analyze_oee,
        predict_maintenance_needs,
        optimize_line_balance
    ]
)
```

6. Orchestrator Agent

python

```python
orchestrator_agent = Agent(
    name="Manufacturing Orchestrator",
    model="gpt-4",
    instructions="""You coordinate between all specialized agents to provide comprehensive insi
    Your role:
    - Route queries to appropriate specialists
    - Synthesize insights from multiple agents
    - Prioritize issues by business impact
    - Generate executive summaries

    Always provide clear, actionable recommendations.""",
    handoffs=[
        supply_chain_agent,
        design_agent,
        quality_agent,
        timeline_agent,
        production_agent
    ]
)
```

📁 Project Structure

```
hardware-digest-system/
├── backend/
│   ├── agents/
│   │   ├── __init__.py
│   │   ├── supply_chain_agent.py
│   │   ├── design_agent.py
│   │   ├── quality_agent.py
│   │   ├── timeline_agent.py
│   │   ├── production_agent.py
│   │   └── orchestrator.py
│   ├── api/
│   │   ├── __init__.py
│   │   ├── main.py              # FastAPI app
│   │   ├── slack_handlers.py    # Slack event handlers
│   │   └── risk_calculator.py   # GTM risk scoring
│   ├── data/
│   │   ├── mock_generator.py    # Dynamic data generation
│   │   └── models.py          # Pydantic models
│   └── utils/
│       ├── redis_client.py
│       └── message_queue.py
├── dashboard/
│   ├── app.py                 # Streamlit main
│   ├── pages/
│   │   ├── 1_📊_Real_Time_Metrics.py
│   │   ├── 2_🎯_GTM_Risk_Score.py
│   │   └── 3_📈_Historical_Analysis.py
│   └── components/
│       ├── metrics_cards.py
│       └── charts.py
├── docker-compose.yml
├── requirements.txt
└── .cursorrules
```

🏃 Day-by-Day Implementation Plan

Day 1: Core Infrastructure & Mock Data

Morning (4 hours)

- Set up project structure
- Configure FastAPI with Slack integration

- Implement Redis for caching/state management
- Create basic message queue with Redis

Afternoon (4 hours)

- Build comprehensive mock data generator
- Create realistic manufacturing scenarios
- Implement data models (Pydantic)
- Test data flow end-to-end

Day 2: AI Agents Implementation

Morning (4 hours)

- Implement all 5 specialized agents
- Create orchestrator agent
- Build agent tools/functions
- Test agent interactions

Afternoon (4 hours)

- Implement GTM risk scoring algorithm
- Create digest generation logic
- Build Slack message formatting
- Test complete agent pipeline

Day 3: Dashboard & Real-time Features

Morning (4 hours)

- Build Streamlit dashboard structure
- Create real-time metrics display
- Implement WebSocket/SSE updates
- Design manufacturing visualizations

Afternoon (4 hours)

- Add interactive features
- Implement drill-down capabilities
- Create alert system

- Polish UI/UX

Day 4: Integration & Demo Prep

Morning (4 hours)

- Full system integration testing
- Performance optimization
- Bug fixes and refinements
- Security hardening

Afternoon (4 hours)

- Prepare demo scenarios
- Create impressive demo data
- Practice presentation flow
- Final polish

💻 Implementation Code

1. FastAPI Main Application

python

```python
# backend/api/main.py
from fastapi import FastAPI, BackgroundTasks, Request, HTTPException
from fastapi.middleware.cors import CORSMiddleware
import redis
import json
from datetime import datetime
import asyncio

app = FastAPI(title="Hardware Digest System")
redis_client = redis.Redis(host='localhost', port=6379, decode_responses=True)

# CORS for dashboard
app.add_middleware(
    CORSMiddleware,
    allow_origins=["http://localhost:8501"],  # Streamlit
    allow_methods=["*"],
    allow_headers=["*"],
)

@app.post("/slack/events")
async def slack_events(request: Request, background_tasks: BackgroundTasks):
    """Handle Slack events"""
    body = await request.json()

    # Slack URL verification
    if body.get("type") == "url_verification":
        return {"challenge": body["challenge"]}

    # Process events asynchronously
    if body.get("type") == "event_callback":
        event = body["event"]
        background_tasks.add_task(process_slack_event, event)

    return {"status": "ok"}

async def process_slack_event(event: dict):
    """Process Slack messages through AI agents"""
    if event["type"] == "message" and not event.get("subtype"):
        # Store in Redis for processing
        message_key = f"message:{event['ts']}"
        redis_client.setex(
            message_key,
            3600,  # 1 hour TTL
            json.dumps({
                "text": event["text"],
                "user": event["user"],
                "channel": event["channel"],
                "timestamp": event["ts"]
            })
        )

        # Trigger agent analysis
        await analyze_message(event)

@app.post("/api/generate-digest")
async def generate_digest(background_tasks: BackgroundTasks):
    """Generate daily digest on demand"""
    background_tasks.add_task(create_and_send_digest)
    return {"status": "Digest generation started"}

@app.get("/api/metrics/realtime")
async def get_realtime_metrics():
    """Get real-time manufacturing metrics"""
    from data.mock_generator import ManufacturingDataGenerator
    generator = ManufacturingDataGenerator()

    return {
        "timestamp": datetime.now().isoformat(),
        "production": generator.get_production_metrics(),
        "quality": generator.get_quality_metrics(),
        "supply_chain": generator.get_supply_chain_status(),
        "gtm_risk_score": calculate_gtm_risk_score()
    }
```

2. Complete Agent Implementation

python

```python
# backend/agents/orchestrator.py
from agents import Agent, Runner, function_tool
from typing import Dict, List, Any
import asyncio

class ManufacturingOrchestrator:
    def __init__(self):
        self.supply_chain_agent = self._create_supply_chain_agent()
        self.design_agent = self._create_design_agent()
        self.quality_agent = self._create_quality_agent()
        self.timeline_agent = self._create_timeline_agent()
        self.production_agent = self._create_production_agent()
        self.orchestrator = self._create_orchestrator()

    def _create_supply_chain_agent(self):
        @function_tool
        def check_inventory_levels(component: str) -> str:
            # Mock implementation - replace with real data
            return f"Component {component}: 15 days coverage, 2 suppliers available"

        @function_tool
        def analyze_supplier_performance() -> str:
            return "Primary supplier: 94% on-time, Secondary: 89% on-time"

        return Agent(
            name="Supply Chain Analyst",
            instructions="""Analyze supply chain risks and opportunities.
            Focus on component availability, supplier performance, and cost optimization.""",
            tools=[check_inventory_levels, analyze_supplier_performance]
        )

    async def analyze_for_digest(self, messages: List[Dict]) -> Dict[str, Any]:
        """Analyze messages and generate digest content"""
        # Categorize messages
        supply_messages = []
        design_messages = []
        quality_messages = []
        timeline_messages = []
        production_messages = []

        for msg in messages:
            text = msg['text'].lower()
            if any(word in text for word in ['supplier', 'component', 'inventory']):
                supply_messages.append(msg)
            elif any(word in text for word in ['design', 'cad', 'bom']):
                design_messages.append(msg)
            elif any(word in text for word in ['defect', 'quality', 'test']):
                quality_messages.append(msg)
            elif any(word in text for word in ['deadline', 'milestone', 'schedule']):
                timeline_messages.append(msg)
            elif any(word in text for word in ['production', 'oee', 'throughput']):
                production_messages.append(msg)

        # Run parallel analysis
        tasks = [
            self._analyze_supply_chain(supply_messages),
            self._analyze_design(design_messages),
            self._analyze_quality(quality_messages),
            self._analyze_timeline(timeline_messages),
            self._analyze_production(production_messages)
        ]

        results = await asyncio.gather(*tasks)

        # Synthesize results
        return self._synthesize_digest(results)
```

3. Mock Data Generator

python

```python
# backend/data/mock_generator.py
import random
from datetime import datetime, timedelta
from typing import Dict, List
import numpy as np

class ManufacturingDataGenerator:
    def __init__(self):
        self.components = [
            "OLED Display 6.8\"", "Snapdragon 8 Gen 3",
            "Camera Module 200MP", "Battery 5000mAh",
            "Memory LPDDR5X 12GB", "Storage UFS 4.0"
        ]
        self.suppliers = ["Samsung", "TSMC", "Sony", "LG Chem", "SK Hynix"]
        self.production_lines = ["Line-A", "Line-B", "Line-C"]

    def generate_slack_messages(self, count: int = 50) -> List[Dict]:
        """Generate realistic Slack messages"""
        message_templates = [
            # Supply chain messages
            "Display supplier reporting {delay} day delay on shipment",
            "Component {component} inventory dropped to {days} days coverage",
            "Urgent: {supplier} quality issue detected in batch {batch}",

            # Design messages
            "Design change request: Antenna placement needs adjustment",
            "BOM update: Switching to {component} for better performance",
            "CAD files updated for camera module housing",

            # Quality messages
            "Defect rate spike to {rate}% on {line} - investigating",
            "Passed drop test certification - 1.5m concrete surface",
            "Quality audit scheduled for {date}",

            # Timeline messages
            "Milestone achieved: EVT build complete",
            "Risk: Regulatory approval may delay by {days} days",
            "Critical path update: Camera module on track",

            # Production messages
            "OEE improved to {oee}% on night shift",
            "Production target achieved: {units} units today",
            "Maintenance required on Line-B SMT machine"
        ]

        messages = []
        for i in range(count):
            template = random.choice(message_templates)
            message = template.format(
                delay=random.randint(1, 5),
                component=random.choice(self.components),
                days=random.randint(5, 30),
                supplier=random.choice(self.suppliers),
                batch=f"B{random.randint(1000, 9999)}",
                rate=round(random.uniform(0.5, 3.0), 1),
                line=random.choice(self.production_lines),
                date=(datetime.now() + timedelta(days=random.randint(1, 7))).strftime("%Y-%m-%
                oee=random.randint(85, 95),
                units=random.randint(8000, 12000)
            )

            messages.append({
                "text": message,
                "user": f"U{random.randint(100, 999)}",
                "channel": random.choice(["supply-chain", "production", "quality", "engineering
                "timestamp": (datetime.now() - timedelta(hours=random.randint(0, 24))).isoforma
                "priority": random.choice(["low", "medium", "high", "critical"])
            })

        return messages

    def get_production_metrics(self) -> Dict:
        """Generate realistic production metrics"""
        hour = datetime.now().hour
        base_rate = 1000 if 6 <= hour <= 22 else 850

        return {
            "current_rate": base_rate + random.randint(-50, 50),
            "daily_production": random.randint(20000, 24000),
            "oee": round(random.uniform(0.85, 0.95) * 100, 1),
            "defect_rate": round(random.uniform(0.01, 0.02) * 100, 2),
            "cycle_time": round(random.uniform(45, 55), 1)
        }
```

4. Streamlit Dashboard

```
python
```

```python
# dashboard/app.py
import streamlit as st
import plotly.graph_objects as go
import plotly.express as px
import requests
import asyncio
from datetime import datetime
import pandas as pd

# Page config
st.set_page_config(
    page_title="Hardware Manufacturing Dashboard",
    page_icon="🏭",
    layout="wide",
    initial_sidebar_state="expanded"
)

# Custom CSS
st.markdown("""
<style>
    .metric-card {
        background-color: #f0f2f6;
        padding: 20px;
        border-radius: 10px;
        text-align: center;
    }
    .alert-box {
        padding: 15px;
        border-radius: 5px;
        margin: 10px 0;
    }
    .critical { background-color: #ffcdd2; }
    .warning { background-color: #fff3cd; }
    .success { background-color: #d4edda; }
</style>
""", unsafe_allow_html=True)

# Initialize session state
if 'auto_refresh' not in st.session_state:
    st.session_state.auto_refresh = True
if 'refresh_interval' not in st.session_state:
    st.session_state.refresh_interval = 30
```

```
# Header
st.title("🏭 Galaxy S26 Ultra - Manufacturing Command Center")
st.markdown("**Real-time monitoring and AI-powered insights for hardware launch success**")

# Top metrics row
col1, col2, col3, col4, col5 = st.columns(5)

# Fetch real-time data
response = requests.get("http://localhost:8000/api/metrics/realtime")
data = response.json()

with col1:
    st.metric(
        "GTM Risk Score",
        f"{data['gtm_risk_score']}/100",
        delta="-3 vs yesterday",
        delta_color="inverse"
    )

with col2:
    st.metric(
        "Daily Production",
        f"{data['production']['daily_production']:,}",
        delta="+2.3%"
    )

with col3:
    st.metric(
        "OEE",
        f"{data['production']['oee']}%",
        delta="+1.2%"
    )

with col4:
    st.metric(
        "Defect Rate",
        f"{data['production']['defect_rate']}%",
        delta="-0.3%",
        delta_color="inverse"
    )

with col5:
    st.metric(
        "Days to Launch",
        "47",
        delta="On Track",
        delta_color="off"
    )

# Main content area
tab1, tab2, tab3, tab4 = st.tabs(["📊 Real-Time", "🎯 GTM Analysis", "📈 Trends", "🤖 AI Ins

with tab1:
    # Real-time production chart
    st.subheader("Production Rate - Last 24 Hours")

    # Generate time series data
    time_points = pd.date_range(end=datetime.now(), periods=96, freq='15min')
    production_data = [random.randint(900, 1100) for _ in range(96)]

    fig = go.Figure()
    fig.add_trace(go.Scatter(
        x=time_points,
        y=production_data,
        mode='lines',
        name='Units/Hour',
        line=dict(color='#1f77b4', width=2)
    ))

    fig.update_layout(
        height=400,
        xaxis_title="Time",
        yaxis_title="Units per Hour",
        hovermode='x unified'
    )

    st.plotly_chart(fig, use_container_width=True)

    # Alert section
    st.subheader("🚨 Active Alerts")

    alerts = [
        ("critical", "Display supplier delay - 3 days impact", "10:32 AM"),
        ("warning", "Camera module defect rate above threshold", "9:15 AM"),
        ("success", "Battery testing completed successfully", "8:45 AM")
    ]

    for level, message, time in alerts:
```

```python
        st.markdown(
            f'<div class="alert-box {level}">⏰ {time} - {message}</div>',
            unsafe_allow_html=True
        )

with tab2:
    st.subheader("GTM Risk Score Breakdown")

    # Risk score components
    risk_components = {
        "Supply Chain": 78,
        "Quality": 82,
        "Timeline": 68,
        "Production": 75,
        "Regulatory": 85
    }

    fig = go.Figure(go.Bar(
        x=list(risk_components.values()),
        y=list(risk_components.keys()),
        orientation='h',
        marker=dict(
            color=['red' if v < 70 else 'yellow' if v < 80 else 'green'
                   for v in risk_components.values()]
        )
    ))

    fig.update_layout(
        height=400,
        xaxis_title="Risk Score",
        xaxis=dict(range=[0, 100])
    )

    st.plotly_chart(fig, use_container_width=True)

# Auto-refresh
if st.session_state.auto_refresh:
    st.empty()
    time.sleep(st.session_state.refresh_interval)
    st.rerun()
```

5. Cursor Rules for Manufacturing Domain

```
# .cursorrules
You are building a manufacturing intelligence system for smartphone production.

## Code Standards
- Use Python 3.10+ with type hints
- Follow PEP 8 style guide
- Add comprehensive docstrings
- Include error handling for all external calls

## Manufacturing Domain Rules
- Production metrics must be realistic (OEE 85-95%, Defect rate 0.5-2%)
- Always use UTC timestamps for global operations
- Implement retry logic for critical operations
- Validate all manufacturing data against reasonable ranges

## AI Agent Guidelines
- Each agent must have clear, specific instructions
- Include domain knowledge in agent prompts
- Implement timeout handling (30s max per agent call)
- Log all agent interactions for debugging

## Testing Requirements
- Create unit tests for all risk calculations
- Include integration tests for agent interactions
- Mock external dependencies properly
- Test with realistic manufacturing scenarios

## Security
- Never log sensitive supplier information
- Implement rate limiting on all endpoints
- Validate Slack signatures on webhooks
- Use environment variables for secrets
```

🚀 Quick Start Commands

```bash
# Clone and setup
git clone <your-repo>
cd hardware-digest-system

# Create virtual environment
python -m venv venv
source venv/bin/activate  # or `venv\Scripts\activate` on Windows

# Install dependencies
pip install -r requirements.txt

# Start Redis
docker run -d -p 6379:6379 redis:alpine

# Start FastAPI backend
cd backend
uvicorn api.main:app --reload --port 8000

# In another terminal, start Streamlit
cd dashboard
streamlit run app.py

# For development with mock data
python backend/data/mock_generator.py --generate-test-data
```

📝 Key Features to Demo

1. **Live Risk Score Updates**: Show how component delays immediately affect GTM score
2. **Multi-Agent Analysis**: Demonstrate different agents analyzing the same issue
3. **Predictive Alerts**: Show AI predicting quality issues before they escalate
4. **Interactive Dashboard**: Drill down from high-level metrics to specific issues
5. **Actionable Digest**: Show how the morning digest drives the day's priorities

---

💡 Pro Tips for Your Demo

1. **Start with Crisis**: Begin demo with a critical alert to grab attention
2. **Show the Solution**: Demonstrate how AI agents identify root cause and suggest fixes
3. **Highlight ROI**: Show time saved (2-3 hours daily) and issues prevented

4. **Make it Interactive**: Let founder ask questions to the AI agents live
5. **End with Success**: Show improved metrics after following AI recommendations

This system will genuinely transform how hardware teams operate, providing the intelligence layer they desperately need for successful product launches!